Acceleration of Parasitic Multistatic Radar System using GPGPU

Prepared By:
Mathew John

Supervised By:
Prof. Michael.R.Inggs

A dissertation submitted to the Department of Electrical Engineering,
University of Cape Town, in fulfilment of the requirements
for the degree of Master of Science in Engineering.

Cape Town, August 2011

Declaration

I know the meaning of plagiarism and declare that all the work in this dissertation, save for that which is properly acknowledged, is my own. This dissertation is being submitted for the degree of Master of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author .....

Cape Town

11 August 2011

Abstract

Parasitic Multistatic Radar (PMR) systems are to equip the world of air traffic surveillance with a reliable and highly cost-effective class of radar systems with counter-stealth abilities. But the computational intensity of the signal processing chain made the process extremely time consuming and acted as the prime hindrance in converting the research project into a practical air surveillance system. Parallel processing using General Purpose Graphic Processing Units (GPGPUs) is used as the solution to handle this computational intensity. The parallel structure of radar signal processing chain with large volume of data fits ideally into the parallel architecture of GPGPUs.

This dissertation details the implementation of PMR signal processing chain in the GPGPU platform. The primary objective of the project is to accelerate the signal processing chain without compromising the algorithm efficiency and to prove that GPGPUs are a promising platform for parasitic radar signal processing. Two distinct clutter cancellation algorithms are implemented together with a high performance matched filter in the GPU platform. The two clutter cancellation algorithm are compared based on their computational and clutter cancellation efficiency and a conclusion about the preferred algorithm for PMR system is made. The GPU implementation of the signal processing chain is compared with the CPU implementation using standard performance metrics focusing on individual stages and the overall system, illustrating the effective acceleration achieved. The dissertation concludes with scope and recommendations for further improvement of the system in a multi-CPU multi-GPU distributed system.

To The Lord God Almighty

And

My Family

Acknowledgements

This research project and dissertation became a reality with the guidance and support of a number of incredible human minds who had always believed in me. First of all, I express my heartfelt gratitude to my supervisor Prof. Michael. R. Inggs who showed me the the light to the Radar Remote Sensing Group and the PMR research project. I am extremely indebted to him for the immense opportunities he had given me through the PMR project, the Netrad trials and funding for the paper presentation at the Signal Processing Symposium-2011, all of which only form an outline in the role he played in my student life. I am thankful to my co-supervisor Dr. Yoann Paichard who introduced me to the world of passive radars.

I would like to extend my gratitude to all the members of RRSG with special thanks to Mr. Gunther Lange and Mr. Craig Tong for the immense help they offered in the test deployment of the system in Western Cape. I am also thankful to Mr. Dario Petri from the University of Pisa for his guidance and the DVB-T data used for the system testing.

There were times in the last two years when my threshold to move ahead had fallen down, and I am extremely thankful to my family who had always stood beside me giving me all the strength and confidence to move ahead. I express my thanks to my friends at Rochester, Ragesh Pillai and my flatmate Thomas who were always more than an extended family to me. I believe all the guidance and support I received, was always an act of God, since the Almighty acts through humans itself. I am indebted to the Lord God Almighty for all the blessings he showered upon me through these marvellous human minds.

Contents

List of Figures

List of Tables

Nomenclature

ARD Amplitude Range Doppler

CFAR Constant False Alarm Rate

CUDA Compute Unified Device Architecture

DPI Direct Path Interference

DSP Digital Signal Processing

ECA Extensive Cancellation Algorithm

FFT Fast Fourier Transform

GPGPUs General Purpose Graphic Processing Units

GPUs Graphic Processing Units

NLMS Normalised Least Mean Square

OPENCL OPEN Computing Language

PGC Professional Graphics Controller

PMR Parasitic Multistatic Radar

SCR Signal to Clutter Ratio

SIMD Single Instruction Multiple Data

Speed- Up Factor The ratio of CPU processing time to GPU Processing time.

Chapter 1

Introduction

The doubling of the number of transistors every 18 months (known as Moore's law [40]) and the demand for sophisticated computer game consoles have come together in the form of Graphic Processing Units (GPUs) and a repackaged version for research computing, known as the General Purpose GPU (GPGPU). GPGPU computing can be a promising technology to many innovative and cost effective engineering solutions. Parasitic Multistatic Radar (PMR) [22] project can be classified as an engineering solution to the need for a new class of reliable, cost-effective radar systems, immune to stealth [28] and will be a solution to the bandwidth congestion all over the world.

This dissertation illustrates the implementation of PMR signal processing on GPUs by exploiting the parallel architecture of GPUs. The aim of this project is to implement a GPU based parallel processing algorithm for PMR system enabling real-time air surveillance. This introductory Chapter begins with a brief insight of PMR, GPUs and the scope and need for parallel processing. At this point, the main objectives of the research project is stated ending up with a short outline of the rest of the dissertation.

1.1 Background

1.1.1 Parasitic Multistatic Radar

The history of radar reveals a technology that was hidden from the electronics world, and that is multistatic radar technology. Multistatic radar [22] refers to a radar system using multiple transmitters and receivers for target detection. Multistatic systems can be both active multistatic radar system using dedicated transmitters, or a Parasitic multistatic system using non-cooperative sources of illumination as transmitters. Figure 1.1 shows the visualisation of a PMR [22] system under implementation for deployment in the Western Cape. The Yellow and Orange arrows represent transmitted waveform from separate transmitters, reflected by target and received by the receivers at University of Cape Town (UCT) and University of Stellenbosch (US) respectively. The distance between the receiver stations is approximately 50 km. The red arrows represent Direct Path Interference (DPI)

[57]. Presently, the PMR system under analysis is a bistatic radar system [57] which means it uses a single set of transmitter and receiver. At this point, a brief history of multistatic radar systems and in particular bistatic radar [22] systems is mentioned.

The first radar experiments in the United Kingdom in 1935 by Robert Watson-Watt demonstrated the principle of radar by detecting a Handley Page Heyford bomber at a distance of 12 km using the British Broadcasting Corporation (BBC) shortwave transmitter at Daventry. Therefore the first radar experiment, and the early 1930s radar experiments were largely bistatic [27]. The British deployed the CHAIN HOME system [42], the French used a bistatic Continuous Wave (CW) radar in a "fence" (or "barrier") system, the Soviet Union deployed a bistatic CW system called the RUS-1 [5], and the Japanese developed a bistatic CW radar simply called "Type A" [56]. The subsequent development of duplexer [54] made the radar world largely monostatic.

In the 1980s, IBM developed a prototype system that could track aircraft in non-real time by analysing the echoes of the vision carrier of an analogue television signal [27]. Lockheed Martin later acquired this division of IBM and developed a commercial PMR system known as Silent Sentry [31]. In 1986, Griffiths and Long [23] published the first open literature work on the utilisation of analogue television signals for radar purposes, but like their predecessors, found that the poor ambiguity function and dynamic range problems prevented them from demonstrating any target detections.

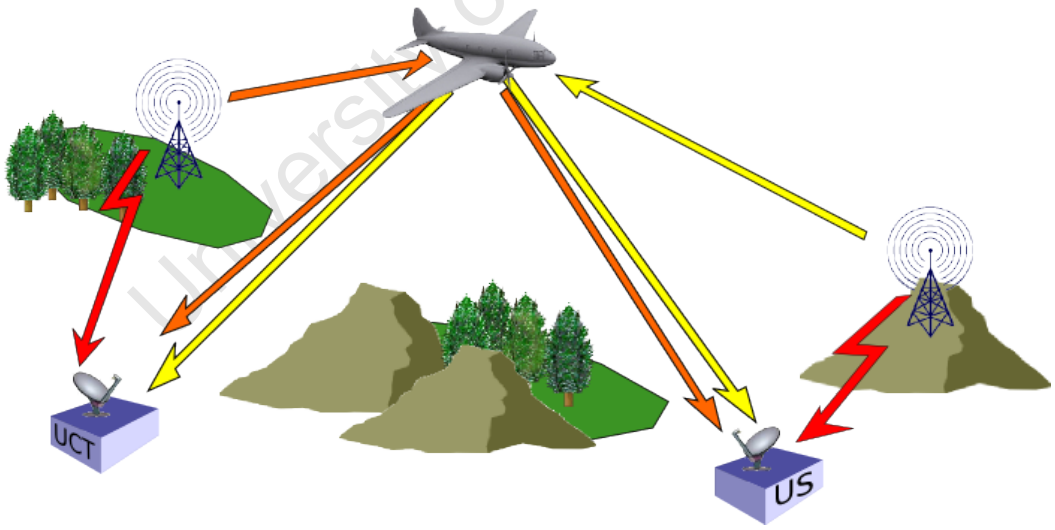

Figure 1.1: PMR System with receiver stations at University of Cape Town (UCT) and University of Stellenbosch (US) [39]

PMR came back into mainstream in mid 1990s. The technology got renewed interest with the arrival of aircraft with stealth capabilities [28]. This renewed interest was probably due to the potential counter-stealth abilities that PMR offered because of its bistatic, low-frequency operation. It was, however, not until the 21st century that PMR started receiving the attention it deserved. The reason for this change in research trend is advancement of Digital Signal Processing (DSP) [10] power and

fast analogue to digital (A/D) conversion. These advances are enabling university research groups to initiate and fulfil PMR projects within reasonable budget and time lines. This also led to a positive impact in the related online literature in open domain.

1.1.2 Graphics Processing Unit

The GPUs are specialised microprocessors for handling 2D/3D graphics. From its first release as IBM Professional Graphics Controller (PGC) [34] in 1984, GPUs were revolutionising the computer graphics industry with their exceptional capability to handle floating point operations. By 2004, a novel concept to harness the massive floating point processing power of GPU came into the world of high performance computing and this is termed as GPGPU.

The Single Instruction Multiple Data (SIMD) architecture [46] of the GPU made it an excellent processor for parallel processing and a new trend of using multi-core GPUs instead of multi-core CPUs is on its way. The GPU market leaders Nvidia and ATI came up with two different programming environment named Compute Unified Device Architecture (CUDA) [46] and OPEN Computing Language (OPENCL)[1] respectively. Due to efficient adaptation from native C language, CUDA and Nvidia GTX 480 FX Fermi graphics card are used for parallel programming in the present project.

1.2 Research Motivation

The literature on PMR systems and the preliminary results from the prototype system in MATLAB provide an insight into the efficiency and reliability of the technology. The technology was lagging behind, due to the high time consumption taken in the signal processing chain. This high processing time could hinder the usage of the the technology as a robust and reliable surveillance system. The need for accelerating the signal processing chain aroused at this stage. In the design of a polyphase filter bank using GPU based on paper by M. Rice [21], the author has investigated the processing power of GPU over CPU. In this scenario, the idea for a new processing code for PMR system based on GPU comes into picture. The scope of this project is to provide evidence that GPUs are a promising computational platform for accelerating PMR signal processing.

The approach used is a step by step migration from CPU to GPU depending on maximum speed-up or acceleration that can be achieved at individual processes. The data used is obtained from direct measurement based on flight data from Cape Town International Airport, University of Pisa and other targets of opportunity.

1.3 Research Objectives

The objectives of this dissertation are as follows:

[1] Accessed 18thDecember 2010<http://www.khronos.org/opencl/>

- Implement two distinct clutter cancellation algorithms in GPU Platform.
- Develop a GPU based high speed matched filter which can be interfaced with Constant False Alarm Rate (CFAR) [54, 43] detection and tracking .
- Compare and benchmark the cancellation algorithms based on time consumption and Signal to Clutter ratio (SCR) [54].
- Compare the GPU and CPU implementations and illustrate the acceleration achieved.
- Draw conclusions about GPU acceleration for radar systems and provide leads for further research and improvement.

1.4 Dissertation Outline

The dissertation in total is composed of 6 chapters. An outline of each chapter is described here:

Chapter 2 concentrates on the core concept of PMR and provide an extensive literature on the topic. The present systems available and a system level description of a complete PMR system is studied. The Chapter is heavily dedicated to signal processing, together with a brief overview of signal acquisition stages. The Chapter also highlights the major processing stages and its importance in the signal processing chain. The Chapter then proceeds to an overview of the GPUs and its adaptation to general purpose programming. The development environment and the toolkit used is mentioned briefly. The scope of using GPUs for PMR system is explained, and concludes with the details of the GPU model used and the Fermi architecture [1].

Chapter 3 concentrates on the PMR system signal processing chain. Based on Chapter 2, the various stages of PMR system is identified in detail. Matched filtering /Amplitude-Range-Doppler (ARD) Processing [57] is identified as an important process, that needs acceleration. The Chapter explains the algorithm model for matched filter and briefly explain the features of the CPU implementation. The Chapter then proceeds to the migration from CPU to GPU. The designing stages and code implementation is explained with reference to flowcharts and the program flow is studied in detail. An initial study on the acceleration achieved, and the processing time for individual processes in the algorithm model, is done at this stage. Clutter cancellation is identified as the next major process in the processing chain. Normalised Least Mean Square (NLMS) [53, 25] algorithm is modelled with respect to the filtering model by Haykin [25]. Salient features of the CPU implementation is studied and then proceeds to the GPU implementation. GPU implementation features and program flow is explained with reference to figures and flowcharts. The Chapter then proceeds to the implementation of performance optimisation measures, recommended by Nvidia [46] for further improvement. Towards the end, the Chapter also gives an insight about the pros and cons of the NLMS algorithm [25] based on results from real world scenario. The GPU implementation was able to achieve an

acceleration of 18.67X for the ARD [57] processing and 8.85X for the NLMS [25] cancellation, resulting in a total speed-up of 16.39X for the PMR signal processing chain in a single GPU. Detailed study on the acceleration achieved will be discussed in Chapter 5. The Chapter concludes with the need for an alternate cancellation algorithm that overcomes the limitations of the NLMS [25].

Chapter 4 is focused on the Extensive Cancellation Algorithm (ECA) [15, 25] which is implemented as an alternative to the NLMS [25] algorithm discussed in Chapter 3. The theoretical aspects of the algorithm is studied with focus on the algorithm complexity and computational requirement prior to code design and implementation. The algorithm model and CPU implementation is mentioned briefly and the individual processes are selected for GPU implementation based on time consumption. The Chapter then proceeds to the detailed implementation of the algorithm in GPU. The implementation of individual processes within the ECA [25] is explained with reference to the flowcharts and hence reveals the heterogeneous nature of the computing platform mentioned in Chapter 2. The ECA implementation on GPU was able to achieve an acceleration of 27.45X, leading to a total acceleration of 29.43X for the PMR signal processing chain in a single GPU. The Chapter concludes with the implementation and performance details of GPU based high performance complex matrix inversion developed for the ECA [25], which can be customised for other DSP applications.

Chapter 5 is dedicated to tests and results. The testing is divided into computational characteristics and clutter cancellation [54] efficiency tests. The computational characteristics and performance measures of GPU implementation of Matched filtering, NLMS [25] algorithm, ECA [15] are studied in detail with respect to the speed-up factor [46] and run-time variation graphs, plotted for different performance parameters. Testing of the clutter cancellation ability of the two algorithms is done with respect to SCR [54] calculation from real world data. ARD plots from different sources of illumination and from simulated data are studied to test the robustness of the algorithms. The Chapter then proceeds to results from test deployment of the system. Multiple target detections are observed at different range and Doppler [57]. The calculated flight path and the actual flight paths are compared. The algorithm efficiency tests in versatile scenarios, studies the properties of both the algorithms in detail. Based on the detailed tests and results, categorised into tests on computational performance and on clutter cancellation efficiency, a conclusion about the preferred algorithm for PMR system is drawn. Though the NLMS [25] algorithm is computationally 10X faster than the ECA [25] , the clutter cancellation efficiency of ECA is superior than the NLMS with reference to the SCR [54] analysis. The speed-up of 27.45X achieved for the ECA [25] in single GPU platform, and the inherent batch process nature [25] of ECA, provides evidence for the effective acceleration the process can achieve in a multi-GPU platform. The Chapter concludes with the selection of ECA as the preferred algorithm for PMR system and validation of the final results by comparing with the CPU results, together with performance details of the complete system.

Chapter 6 as the final Chapter depicts the conclusions drawn from the GPU implementation of PMR signal processing with reference to the level of acceleration achieved and the test results from Chapter 5.

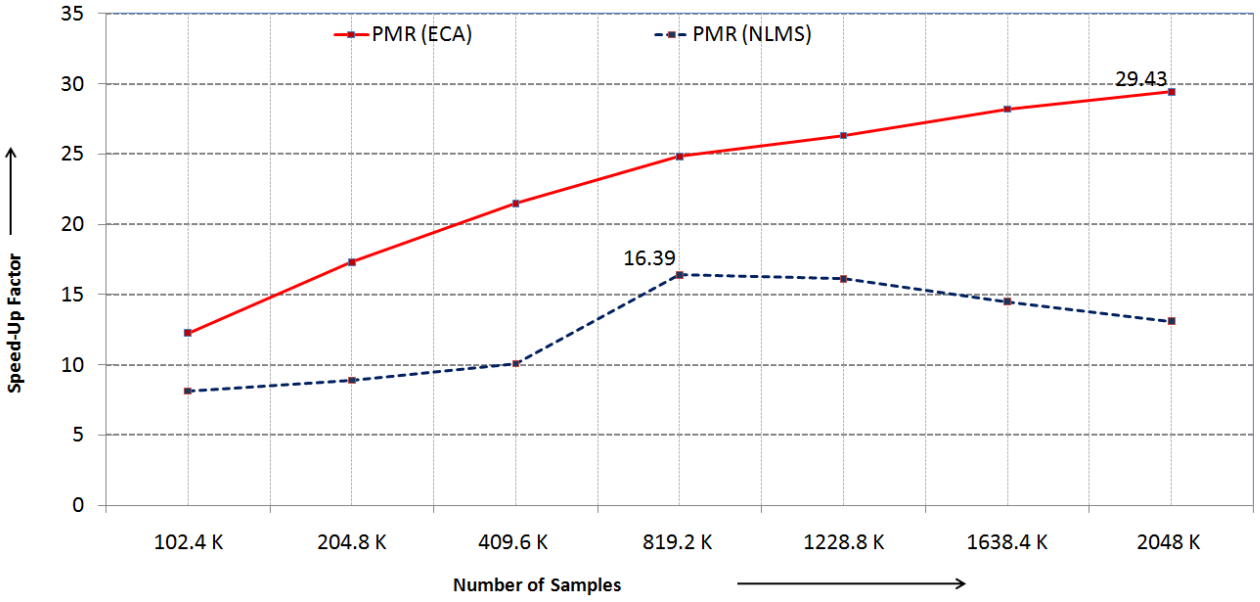

Figure 1.2: Overall Speed-Up Factor for the PMR system. (The processing is done using ECA with cancellation for 300 Range bins, Doppler bins=3 and ARD plot for 300 Range bins and 1201 Doppler bins centred at 0 Hz.)

The acceleration achieved for the signal processing chain in a single GPU platform, providing evidence for real time processing in a multi-GPU platform is mentioned. The recommendation for multi-GPU implementation, including use of multi-threaded CPU for highly optimised heterogeneous platform is discussed. The Chapter also mentions the future work on this project focused on interfacing the code to streamline plotter [55] and CFAR detector [54]. The Chapter concludes with scope and leads for further research and recommendations for improvement of the system.

Chapter 2

PMR and GPGPUs

The ever increasing air traffic congestion all over the World requires multiple surveillance and control systems forming a sophisticated network of radars. The cost of design, implementation and maintenance of such a surveillance network using conventional active radars, is not in the reach of the developing economies in Asia and Africa. The requirement for such a network is inevitable in South Africa, Middle East and Far East Asia, which acts as new hubs for international air travel with changing trade and economic patterns. Another supporting factor for an alternative from conventional radars is the exponentially increasing frequency spectrum congestion. The spectrum congestion demands the exploitation of the same frequency band for multiple use. The role of PMR [22] together with its anti-stealth [28] abilities, is thus gaining importance in this scenario.

PMR can be classified under Bistatic Radar [9] technology which utilises separate transmitter and receiver instead of duplexer. PMR technology deviates from traditional bistatic radars in the case of transmitters. Instead of using a dedicated transmitter for the source signal, PMR uses emissions from sources of illumination which are already present in the environment. The source of illumination can be any existing transmitter like FM Radio [50], Global System for Mobile communications (GSM) signals [17], Digital Video Broadcast-Terrestrial (DVB-T) [18] and Digital Audio Broadcasts (DAB) [18].

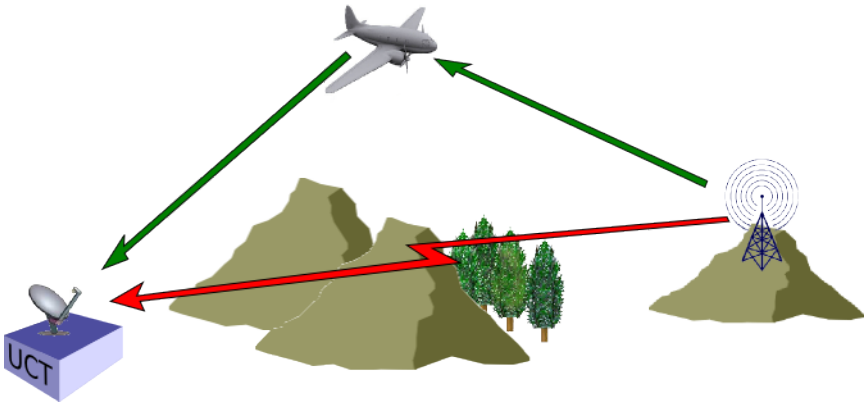

Figure 2.1: System Geometry of a typical PMR in Bistatic Configuration [39]

Since PMR uses illuminators of opportunity, transmitter parameters and characteristics of the transmitted waveform are not under the control of the the radar design engineer. The exploitation of a non-dedicated waveform necessitates the need for a complex processing algorithm. A preliminary PMR signal processing algorithm [26] was designed at Radar Remote Sensing Group (RRSG) at the University of Cape Town. Processing of massive amount of data in this computationally intensive algorithm made the process extremely time consuming. This large computation time limits the technology from using as a reliable air surveillance system. The need for a new processing algorithm which is computationally efficient comes at this stage. The parallel nature of the processing algorithm suits for processing using GPGPUs.

GPUs which are specialised microprocessors to accelerate graphics rendering [48] is the backbone of high end gaming and visualisation industry. The highly parallel nature of GPUs, is dedicated for graphics rendering [48] thereby reducing the load on Central processing unit (CPU). The capability of GPU to perform millions of floating point operation per second can be used for general purpose programming by addition of programmable stages and high precision arithmetic, to the rendering pipelines [33]. Thus a relatively new concept of computing using GPGPU or GP^{2}U comes into existence. The evolution of GPUs to GPGPUs is explained in detail in [33].

This Chapter begins with explanation of the core concepts of PMR, giving more importance to the signal processing chain. The architecture of the PMR system is studied with brief explanation of individual stages. The stages selected for GPU computation are explained in detail. At a later stage, the Chapter deals with the internal architecture of GPGPUs and CUDA as the processing environment. The Chapter concludes with an overview of the Nvidia GTX 480 FX series [47], which is the GPU used for the algorithm implementation.

2.1 Architecture of PMR System

The architecture of the PMR system in bistatic configuration, is shown in Figure 2.2. Figure 2.2 illustrates the different stages of processing and highlights the computation platform used for each stage.

2.1.1 Data Acquisition

RF Front-end

The data acquisition begins with the RF Front end consisting of a Yagi-Uda antenna [12] as the reference antenna which is directed towards the the transmitter and a Log Periodic Dipole Antenna (LPDA) [12] as the surveillance antenna directed towards the coverage area. The antennas have main-lobe gain of 10 and 14 for the reference and surveillance channels, respectively.

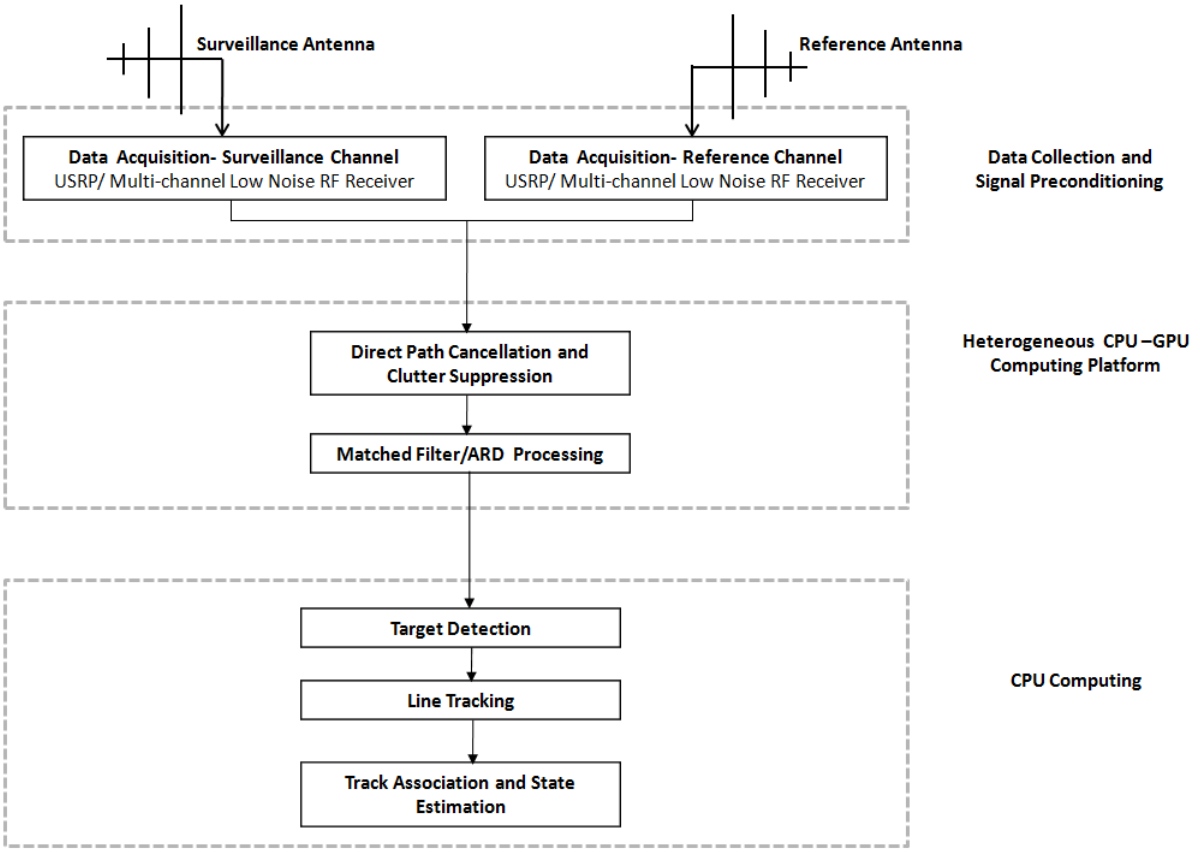

Figure 2.2: Architecture of the PMR System

Analogue/Digital (A/D) Converter and Front end noise removal

A/D conversion and front end noise removal can be done in two discrete ways. The earlier work in this field used the Universal Software Radio Peripheral (USRP) [19, 26]. USRP [19] board in conjunction with GNU Radio[1] software, was used to convert the analogue signal from reference and surveillance/echo channel into a single stream of data at 32 MB/s after decimation. The USRP [19] output is connected to signal Processing PC via USB 2.0. USRP [19] can be used as a cost-effective digitisation solution. However, the PMR system under implementation exploits FM channels with a Bandwidth of 200 kHz. Digitisation at this data rate results in USB buffer overrun. This problem is solved by using a dedicated multichannel Low noise receiver for data acquisition. The receiver used is capable of digitising sharply filtered band in a range of 20-3600 MHz. The receiver features minimum dynamic range of 75 dBm and image rejection of 85 dB. The use of this receiver as a reliable and robust RF front end, and digitisation solution is thus preferred for the PMR project at this stage.

[1] Accessed 12 November 2010, <http://gnuradio.org>

2.1.2 Direct Path Interference Cancellation and Clutter Suppression

Figure 2.1 illustrates the presence of DPI [57] in the surveillance channel at the RF Front end. This direct path interference is handled to an extend by physical and spatial techniques. But for reliable operation, DPI cancellation is done as a major process in signal processing.

Clutter suppression, as in any radar system, is the the removal of time-delayed reflections from unwanted targets, ground and atmosphere. Two different algorithms were developed for direct signal cancellation and clutter suppression-

- NLMS Algorithm [25].
- ECA [25, 15].

Due to computational complexity, CPU version of both algorithms has very high processing times. The need for the speed accelerated GPU version aroused at this stage and the work of the author focuses on accelerating the signal processing chain using GPU. Chapter 3 and Chapter 4 concentrates more on the modelling and implementation of both algorithms. A comparative study of both the algorithms based on tests and results obtained, is illustrated in Chapter 5.

2.1.3 Matched filtering

Matched filtering is the stage of PMR signal processing where target detection is obtained in the ARD [57] plot. This stage can be considered as the search for the time-delayed and Doppler-shifted versions of the reference signal. This is achieved by correlating the surveillance signal with Doppler-shifted versions of the reference signal to form a bank of filters matched to every possible Doppler frequency of interest.

This is equivalent to calculating the ambiguity function [57] and can be written as follows:

$$\left|\Psi(R_R, f_d)^2\right| = \left| \int_{-\infty}^{\infty} e(t) d^*(t + R_R) e^{j2\pi f_d t} dt \right|^2 \tag{2.1}$$

where $\left|\Psi(R_R, f_d)^2\right|$ denotes the ARD surface being calculated, $e(t)$ represents the surveillance signal, $d(t)$ represents the reference signal, R_R represents the range of interest and f_d represents the Doppler shift of interest.

The CPU algorithm for matched filtering was based on calculating high resolution Fast Fourier Transform(FFT) using the FFTW[2] library. Development of GPU version is illustrated in Chapter 3. Timing and performance comparison of CPU and GPU matched filtering algorithms is illustrated in Chapter 5.

[2] Accessed 15 December 2010 , http://www.fftw.org/#documentation

A decimated version of the ARD algorithm using Cascaded Integrator Comb (CIC) [51] filters was also developed in C language and the GPU version for the same was developed. But the GPU version of the non-decimated version gave better results in terms of both speed-up and overall- runtime than the decimated version. Since decimation using the CIC filter is a sequential process, and when the data size increases with increase in time of observation, the processing time for filtering also increases. This results in the non-decimated version of ARD algorithm to perform better in the GPU platform with higher effective speed-up and lower processing time than the decimated version.

2.1.4 Target Detection

Targets are detected on the calculated ARD surface [57] by applying an adaptive threshold [43]. Range and Doppler [54] cells that exceeds the detection threshold are identified as targets. The detection threshold is varied as an estimate of the noise variance. A standard cell averaging constant false alarm rate (CFAR) algorithm [43, 54] is used.

2.1.5 Line Tracking

The output of the CFAR algorithm [54] produces all the cells on the ARD surface that contain target detections. It is now necessary to associate this range and Doppler data with individual targets. A standard Kalman filter [24] can be used to effectively track targets in the Range-Doppler space [9]. Most false alarms are rejected during this stage of the processing.

2.1.6 Track Association and State Estimation

When multiple transmitters are used, a target can be potentially detected by every transmitter. The return from this target will appear at a different bistatic range [57] and Doppler shift [57] with each transmitter. Hence it is necessary to determine which target-returns from one transmitter, correspond with those on the other transmitters. Having associated these returns, the point at which the bistatic range ellipses from each transmitter intersect is the location of the target. The optimum approach is to combine the measurements from each transmitter using a non-linear filter, such as Kalman filter [24]. The Kalman filter [24] can be also utilised to estimate the state of the target including location, heading and speed from the full measurement set of bistatic range, bearing and Doppler.

2.2 GPGPU: An Introduction

Computer Graphics is a technology based industry which has witnessed a revolutionary increase from a few pixel based 2D games to the world of high performance 3D gaming and visualisations back boned by billions of pixels. The reason for this dramatic increase is the evolution from on-board graphic rendering microprocessor to dedicated Ultra Large Scale Integration (ULSI) based GPUs.

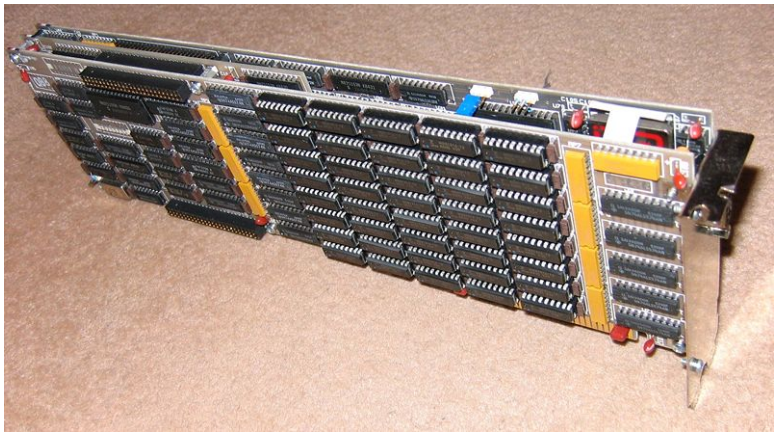

Figure 2.3: IBM Professional Graphics Controller-:The first 2D/3D Graphics accelerator [34]

Figure 2.3 shows the first 2D/3D Graphics accelerator, the IBM Professional Graphics Controller (PGC) [34] released in 1984. It was very advanced for the 1980s, but continuous research in graphics processing and the need by computer visualisation industry helped Nvidia to release the World's first GPU, the Geforce 256 (also known as NV10) shown in Figure 2.4.

The ability of GPUs to handle high intense arithmetic operations and its parallel architecture led to the introduction of the novel concept of using them for General Purpose computing. Thus GPU revolutionised itself to become GPGPU or GP^2U. Subsection 2.2.1 on the following page introduces the features of GPGPU, as efficient parallel processors.

Figure 2.4: Nvidia Geforce256 [16]

2.2.1 GPGPUs-An efficient Parallel Processor

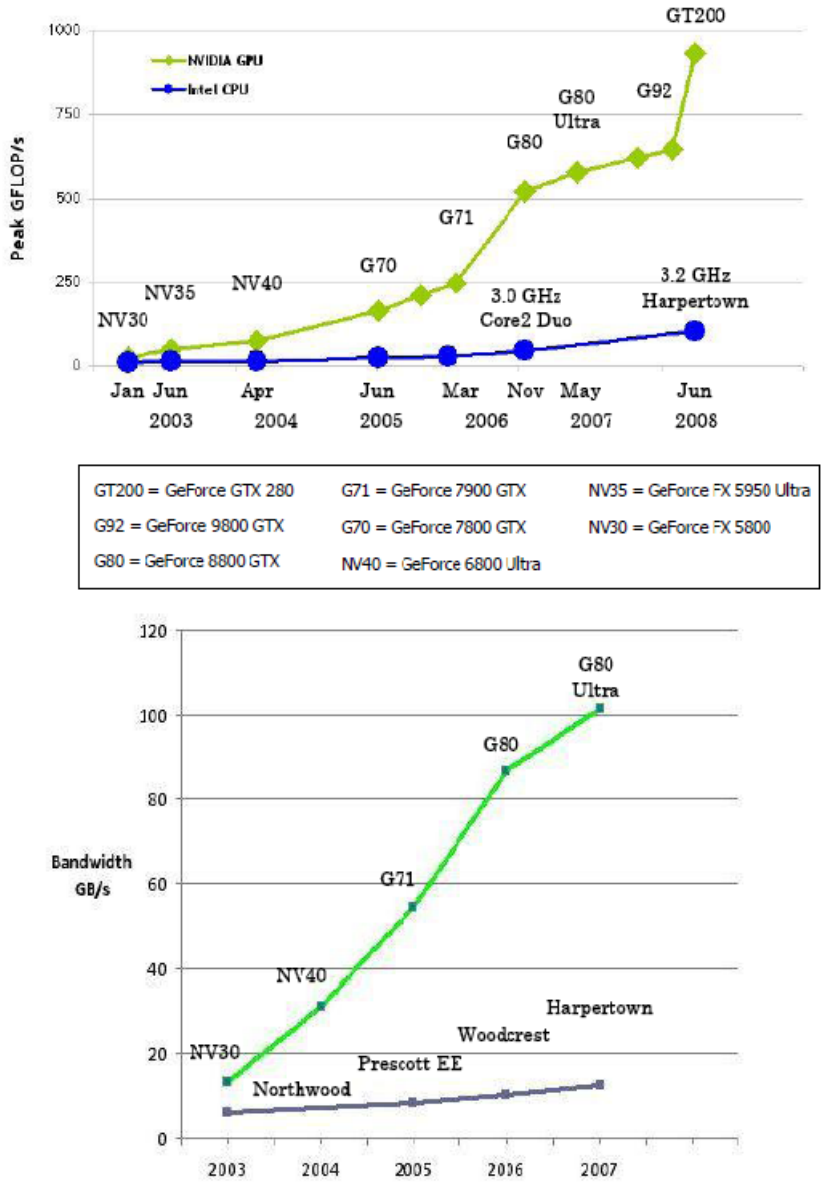

Figure 2.5: Floating-Point Operations per Second and Memory Bandwidth for the CPU and GPU [46]

GPUs are efficient than CPUs for compute-intensive, highly parallel computation since they are fundamentally designed for graphic rendering where large sets of pixels and vertices [33] are mapped for parallel processing. The architectural fact for this performance supremacy is due to the reason that more transistors are devoted to data processing, rather than data caching and flow control. Figure 2.5 illustrates a comparison between floating point operation per Second and memory bandwidth for the CPU and the GPU.

GPUs are specialised for applications that fall in the following criteria:

- Data parallel Computation

  Applications in which the same program is executed on many data elements in parallel and the amount of data handled is huge. Figure 2.6 illustrates that GPU devotes more transistors for data processing than data caching and flow control.

- Applications with high arithmetic intensity

  Arithmetic intensity refers to the ratio of arithmetic to memory operations. Frequent memory operations require efficient flow control in which heterogeneous programming mentioned in subsection 2.2.2 becomes necessary.

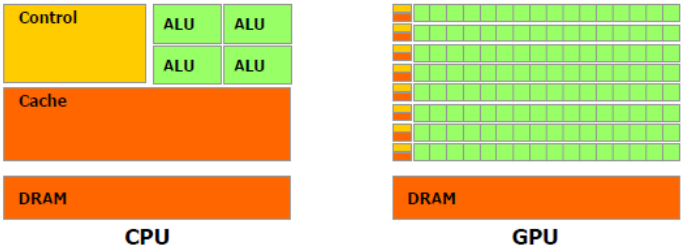

Figure 2.6: The GPU Devotes More Transistors to Data Processing [46]

Applications with the above characteristics has lower requirement for sophisticated flow control. Because the same program is executed on many data elements and due to high arithmetic intensity, the memory access latency can be hidden with calculation instead of big data caches.

2.2.2 CUDA

Nvidia introduced CUDA [46] in late November 2006 as a general purpose parallel computing architecture. CUDA is developed as an extension of ANSI C [11] helping developers for a smooth migration to GPU programming.

Scalable programming model

The parallel programming model characterised by multi-core CPUs and GPUs and the applications developed based on it should scale with Moore's law [40]. The application should scale themselves with increasing number of cores in future systems. and CUDA handles this crucial design requirement by maintaining a simple but efficient architectural hierarchy. The programme is partitioned at core level into threads and a group of threads form a block. Thus a multi-threaded CUDA program can be illustrated as a set of blocks. Depending upon the number of cores present in the GPUs, the blocks can be scaled. Thus the programme functionalities which are taken care at thread level are independent with the number of cores in the GPU used. This unique design, scales CUDA programs

to future machines with higher compute capability. The block level architecture and the automatic scaling of blocks depending upon the number of cores in the device is shown in Figure 2.7. From Figure 2.7, it is evident that a GPU with more cores will execute the program in less time than a GPU with fewer cores.

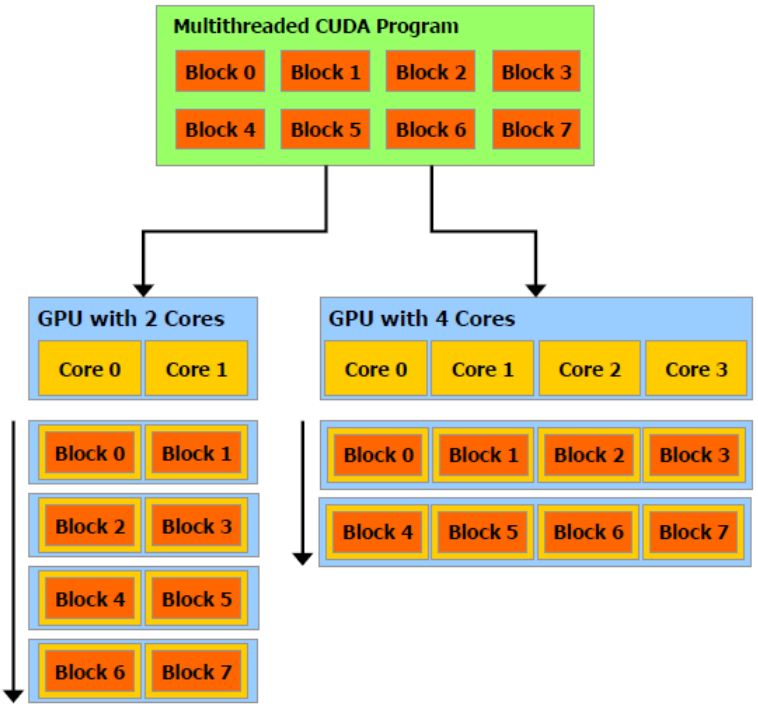

Figure 2.7: Scaling of blocks into different devices depending on number of cores [46]

Heterogeneous programming Environment

Heterogeneous programming previously mentioned can be best exercised since CUDA can be considered as a subset of ANSI C. Commercial and scientific projects require both sequential and parallel processing, since I/O operations and inherently sequential functionalities are inevitable in most high end applications. At designing stage the developer can switch between standard C and CUDA for serial and parallel execution demanding upon required functionalities. In CUDA programming environment, the CPU is termed as host and the GPU is termed as device. The relationship between the host and the device has many similarities to a master slave relationship in computer networks. The compute intensive application that fall in the criteria mentioned in subsection 2.2.1 will be sent from host (CPU) to the device (GPU). After the processing in the device, the data is sent back to the the host for continuing serial execution and other I/O operations.

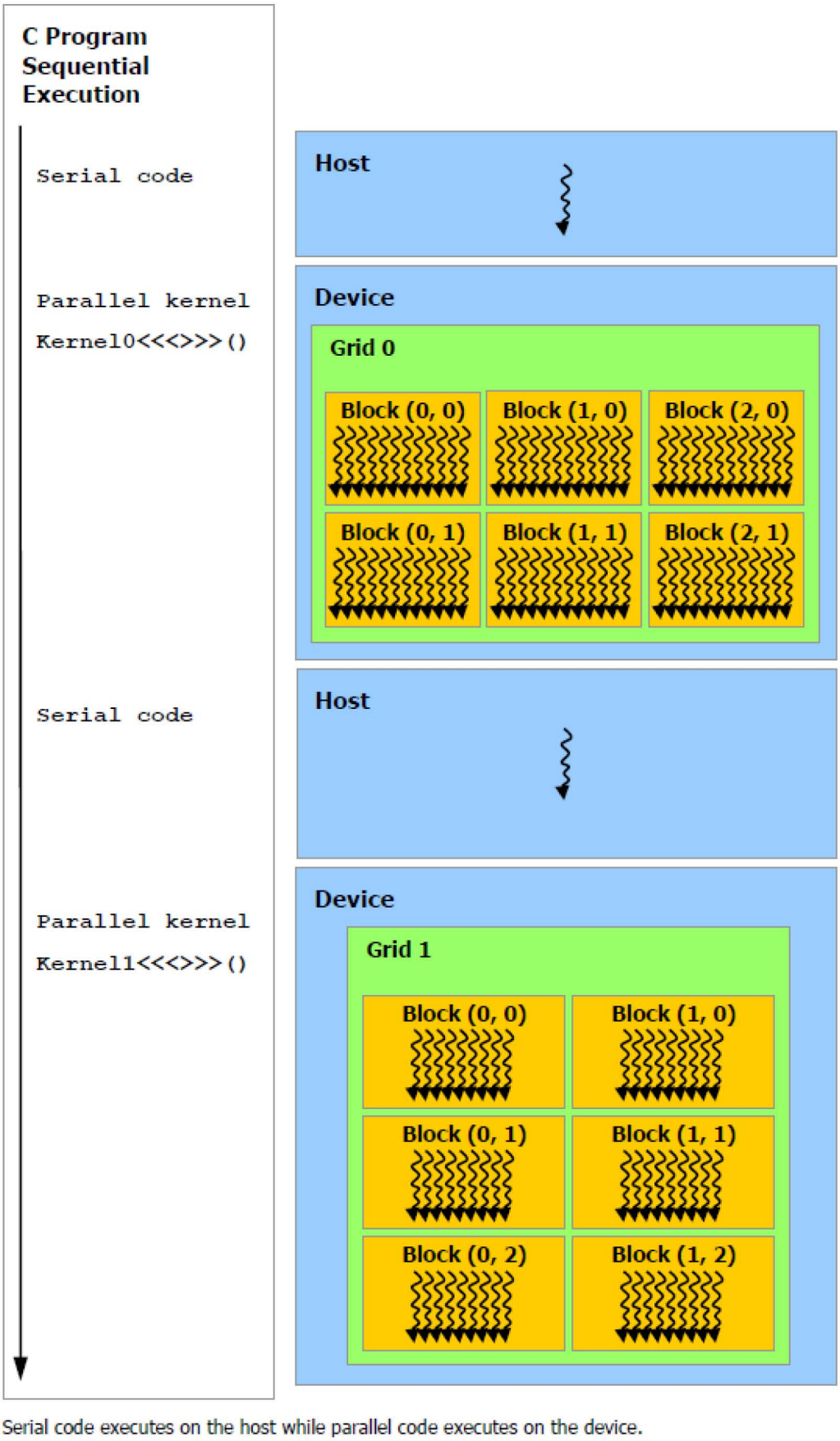

Figure 2.8: Heterogeneous Programming Model [46]

Figure 2.8 illustrates the concept of heterogeneous programming and the data flow between host and device. The data flow between the host and the device is via PCI express bus [30]. The situation leverages memory bandwidth as a crucial factor in GPU based heterogeneous programming environment.

2.2.3 CUDA Programming Model

Terminologies and Parameters

- Kernels: Functions defined to run on the device are called kernels. A kernel is defined using the _global_ declaration specifier and the number of CUDA threads that execute for a given kernel call is specified using <<<...>>> execution configuration syntax .
- Thread: The smallest level of program execution on each parallel path can be considered as a thread. Each thread has a unique thread ID that is accessible within the kernel using the built-in threadIdx variable.
- Thread Block/Block: threadIdx is a 3-component vector, so each thread can be identified when it constitutes a 1 dimensional to 3 dimensional thread blocks. There is a limit to the number of threads per block depending on the device. Currently, a thread block may contain up to 1024 threads.
- Grids: Blocks are organised into grids which can be one dimensional or two dimensional depending on application. The number of thread blocks in a grid is governed by the size of data that have to be handled by the application and the number of processors in the system.

The number of threads per block and the number of blocks per grid is specified using execution configuration syntax, during a kernel call. Figure 2.9 illustrates the arrangement of threads, blocks and grids.

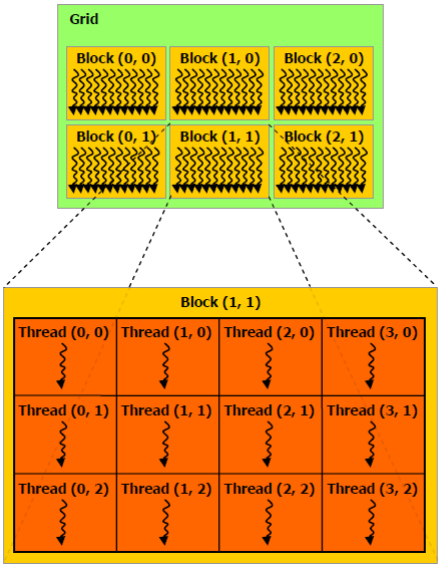

Figure 2.9: Grid of Thread Blocks [46]

Memory Hierarchy

CUDA threads may access data from multiple memory spaces during their execution as illustrated by Figure 2.10. Each thread has private local memory. Each thread block has shared memory visible to all threads of the block and with the same lifetime as the block. All threads have access to the same global memory. There are also two additional read-only memory spaces accessible by all threads: the constant and texture memory spaces.

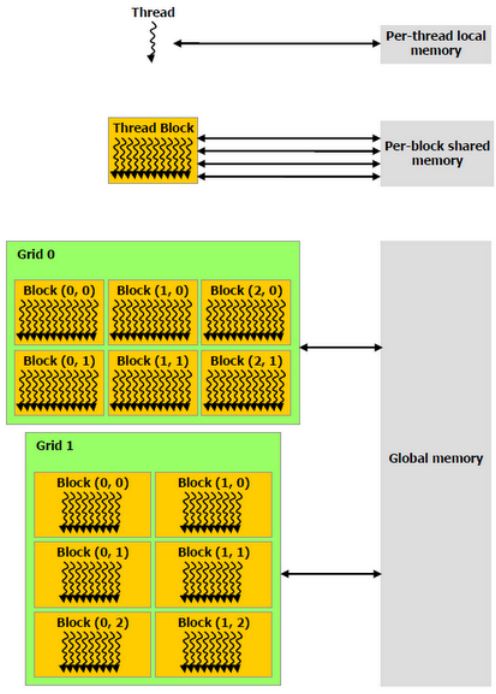

Figure 2.10: Memory Hierarchy [46]

The global, constant, and texture memory spaces are optimised for different memory usages. Texture memory also offers different addressing modes, as well as data filtering, for some specific data formats. The global, constant, and texture memory spaces are persistent across kernel launches by the same application.

2.2.4 Fermi series and GTX 480FX

The device used for the implementation is GTX 480 FX, which is categorised in the Nvidia Fermi series [1]. Fermi architecture is considered as a major breakthrough in GPU computation. The architecture is more suited for double precision arithmetic operations with 32 CUDA processors, with each processor having 15 streaming processors resulting in a total of 480 high performance CUDA cores. Detailed device level specification is added in the Appendix B. According to Nvidia, the Fermi series Nvidia Tesla 20-series GPGPU processors deliver equivalent performance to a quad-core CPU at 1/10th the cost and 1/20th the power consumption [2]. Thus in addition to processing speed-up, the use of Fermi series [1] GPU platform, will help to reduce the computational cost of the PMR project, without compromising in computational performance and algorithm efficiency which will be revealed by the following Chapters.

2.3 Conclusion

This Chapter provided an insight into PMR system and highlighted the major stages in the PMR signal processing. The architecture and signal processing stages of PMR in bistatic configuration were discussed. The Chapter also introduced GPGPU as an efficient platform for parallel computation. The architecture of GPUs and CUDA programming model were discussed. The Chapter also introduced basic features of Fermi series Nvidia GTX 480 FX as the GPU used for parallel computing. The need for accelerating the signal processing chain using GPU was identified. In short this Chapter basically introduced two technologies- PMR and GPGPU, which are integrated together in Chapter 3.

Chapter 3

GPU Implementation of PMR Signal Processing

The introduction to the PMR system and GPUs, as a general purpose parallel computing platform, was discussed in Chapter 2. This Chapter explains the design and implementation of a CUDA based parallel processing algorithm for PMR signal processing. The architecture of PMR, in Section 2.1 illustrated the major phases in the signal processing chain. Since the primary aim of GPU implementation is to speed up the processing chain, individual processes were identified based on their computational intensity and nature of the algorithm. The following processes were selected for GPU implementation based on the fact that both of them satisfy the conditions for efficient parallel processing mentioned in Section 2.2.1:

- DPI and Clutter Cancellation.
- Matched filtering.

Matched filtering was identified with more importance since the CPU based sequential algorithm was extremely time consuming and the proposed GPU implementation can be used to generate ARDs [57] for other bistatic radar systems [57] also. The Chapter begins with a brief theoretical background of matched filtering and proceeds to algorithm modelling. The Chapter then gives an insight about the present CPU implementation and extends to the GPU implementation.

The second half of the Chapter deals with DPI and Clutter Cancellation. NLMS [25] algorithm mentioned in subsection 2.1.2 is modelled here and proceeds to the step by step migration from CPU to GPU. The Chapter concludes with the pros and cons of NLMS [25] algorithm, necessitating the need for an alternate DPI and clutter cancellation algorithm.

3.1 Model for Matched Filtering

The concept of Matched filtering was introduced in subsection 2.1.3. Equation 2.1 on page 11 can be written in discrete time domain [10] as follows:

$$\left|\Psi(\tau,\nu)^2\right| = \left|\sum_{n=0}^{N-1} e(n)d^*(n-\tau)e^{j2\pi\nu n/N}\right| \tag{3.1}$$

[26, 57]

where $\left|\Psi(\tau,\nu)^2\right|$denotes the ARD surface,$\Psi$ denotes the time-delay of interest and ν denotes the Doppler-shift of interest [50].

Algorithm Modelling

The basic steps that have to be performed for ARD [57] calculation, based on equation (3.1), are as follows:

1. Take N samples of the direct signal d(n), delay it by rotation depending upon total number of delays and conjugate it to obtain $d^*(n-\tau)$.
2. Calculate the dot-product of $d^*(n-\tau)$ and the echo/surveillance channel signal $e(n)$ by an iteration statement limiting to the size of the data-set .
3. Use the Fast Fourier transform (FFT) [10]on the dot product for conversion to frequency domain.
4. Discard the FFT bins that are not of interest specified by an iteration statement limited to the maximum number of Doppler [57].
5. Repeat the above steps to the maximum bistatic delay or range [57] specified.
6. Find the absolute value of the complex ARD and return it together with the complex ARD value, $\left|\Psi(\tau,\nu)^2\right|$ mentioned in equation (3.1), and return the value to the main program .

Algorithm Requirements

1. The data size of the input depends on sampling frequency and the time of observation. The test data has a sampling frequency of 409.6K Samples/second. Typical time of observation extends up to 5 seconds making the input data size reach the figure of 2M Samples/data set. The memory allocation should be able to handle this input size. Priority must be given to memory deallocation after use.
2. The Algorithm should be written as a calling function from the main program, but should be easily customisable as a standalone programme in order to use in zero clutter simulations, like the Flexible Extensible Radar Simulator (FERS) [36].
3. The number of Doppler bins and bistatic delay [57] should be variable using command line, so that the ARD surface can be obtained for the preferred region of observation.

4. The algorithm should have two return values-

   (a) The ARD value as a complex variable for CFAR [43] interface.

   (b) The absolute value of ARD for the streamline plotter [55].

3.1.1 CPU Implementation of Matched Filtering

Matched Filtering was first developed for simulation purpose in MATLAB as a part of previous work [26] done in this field. The CPU implementation is adapted from this and serves as the benchmark for comparison with GPU version. The CPU version is written in C language [11]. The salient features of the CPU code are:

- The user specifies the size of the data set, and the number of Doppler and bistatic delay in the main program which is then passed as variables for ARD calculation function.
- The delayed version of the direct signal is obtained, and stored in dynamically memory allocated temporary variables, and rotated to get the conjugated version.
- The dot product of the conjugated direct signal and echo signal is calculated and stored in the input variable for FFT [10] calculation in complex float type.
- The FFTW [38] library is used to perform the Fast Fourier transform [10]. One dimensional complex-to-complex forward FFT "plan" was created using FFTW [38], which is called when the operation needed to be run on the data. Plans are used in FFTW [38] to define an optimised algorithm for a particular transform given the FFT size, input and output arrays, and the direction of the FFT (forward or inverse) are mentioned. When the plan is created, the program searches through a set of possible FFT parameters for the values that provide maximum performance, and is of most use when calling the same FFT multiple times. The time spent searching through the problem space for an optimal plan can be controlled using the flags, FFTW_ESTIMATE [38] for a quick search or FFTW_MEASURE [38] for a longer, more comprehensive one. FFTW_ESTIMATE [38] was used for this implementation.
- The FFTW [38] library was configured to use single-precision floating-point numbers, by using the –enable-float flag when compiling it. Then all variables were declared using the fftw_complex type. The FFTW [38] used an outplace transform since separate variables were used for input and output.
- Selection of desired Doppler and range is performed on the FFT output so that only interested area of coverage is constituted in the ARD surface.
- As mentioned in the algorithm requirement, the absolute value of the ARD is calculated which acts as the input to the streamline plotter and the complex value is used for CFAR [43] detection.

3.1.2 GPU migration

The implementation of algorithm in GPU can be better termed as 'migration' rather than implementation since the GPU algorithm is a step by step transformation of the CPU algorithm. To serve the primary purpose of computational speed up, check points to calculate processing time for individual process in CPU was done. The CPU and GPU timing for individual processes is illustrated in Table 3.1. Table 3.1 reveals that FFT calculation as the major process for GPU conversion followed by absolute value and dot product calculation. Flowchart illustrating the computation, with data and control movement in the CPU-GPU heterogeneous platform is shown in Figure 3.1. The salient features of the GPU implementation are as follows:

1. Basic Steps

   (a) Initialise the preferred GPU device and get the device properties.

   (b) The data received from the main program is in the CPU as host variables [46]. cudamemcopy [48] from host to device is used to transfer the data from CPU to GPU for calculation.

   (c) The output after parallel processing in the device is copied back to host using cudamemcopy from device to host.

2. Dot product using custom kernel

   (a) The conjugated version of the direct signal and echo signal is copied from host to device variables.

   (b) The number of threads per block and blocks per gird are set depending upon GPU initialised using execution syntax [46].

   (c) The GPU kernel set as a _global_ function [48] is called, and the vector multiplication is executed in parallel threads in the GPU.

   (d) The dot product is retained in the GPU for FFT processing.

3. FFT using CUFFT

   (a) CUFFT [44] is a CUDA library and is used to calculate the FFT. CUFFT [44] is developed based on FFTW library and has got a similar plan structure as FFTW [38].

   (b) The FFT input and output are declared in cufftcomplex data type [44].

   (c) The parameters in the CUFFT plan [44] are FFT size depending upon FFT resolution [10] required, complex to complex, 1-Dimensional FFT [44].

   (d) When the function is called, the plan is executed and the output is obtained in the device variable.

4. ARD absolute value using custom kernel

   (a) The process is similar to dot product calculation already mentioned with change in the kernel functionality.

Process	CPU Timing (sec)	GPU Timing (sec)	Speed-up Factor
Dot Product Calculation	2.24	0.31	7.22X
FFT Calculation	14.30	0.85	16.82X
Absolute Value Calculation	3.16	0.38	8.31X
Total Processing Time	19.70	1.54	12.79X

Table 3.1: Comparison of CPU vs GPU time consumption for individual processes in matched filtering for Data-size=819.2 K Samples, Range bins=100 and Doppler bins =500

3.2 DPI Cancellation and Clutter Suppression

DPI and Clutter Cancellation was introduced in subsection 2.1.2 with reference to Figure 2.1. Correlation of these unwanted reflections (clutter) from stationary sources [25] with the reference signal leads to:

- Strong clutter echoes masking targets with high Doppler frequencies.
- A fraction of the direct signal received via the side/backlobe of the surveillance antenna masks target echo signals.
- Strong target echoes masking lower power echoes coming from other targets, even in the presence of large range-Doppler separations.

A reliable and robust radar system needs a dedicated signal processing on the captured data, to effectively control the effects of clutter listed above. Adaptive filtering [25] on the target channel is included as a signal processing stage, before applying matched filtering for this purpose.

3.2.1 Adaptive Filtering

The signal model at the receiver can be analysed for adaptive filtering by Haykin [25] as:

$$s_R(n) = A(n)\, s_T(n) + \sum_{k=0}^{N_T} a_k s_T(n-\tau_k)\, e^{j2\pi f_{dk} n T_c} + \sum_{i=1}^{N_c} c_i s_T(n-\tau_i) + v_R(n) \tag{3.2}$$

where

- $s_T(n)$is a sample of the complex envelope of the reference waveform

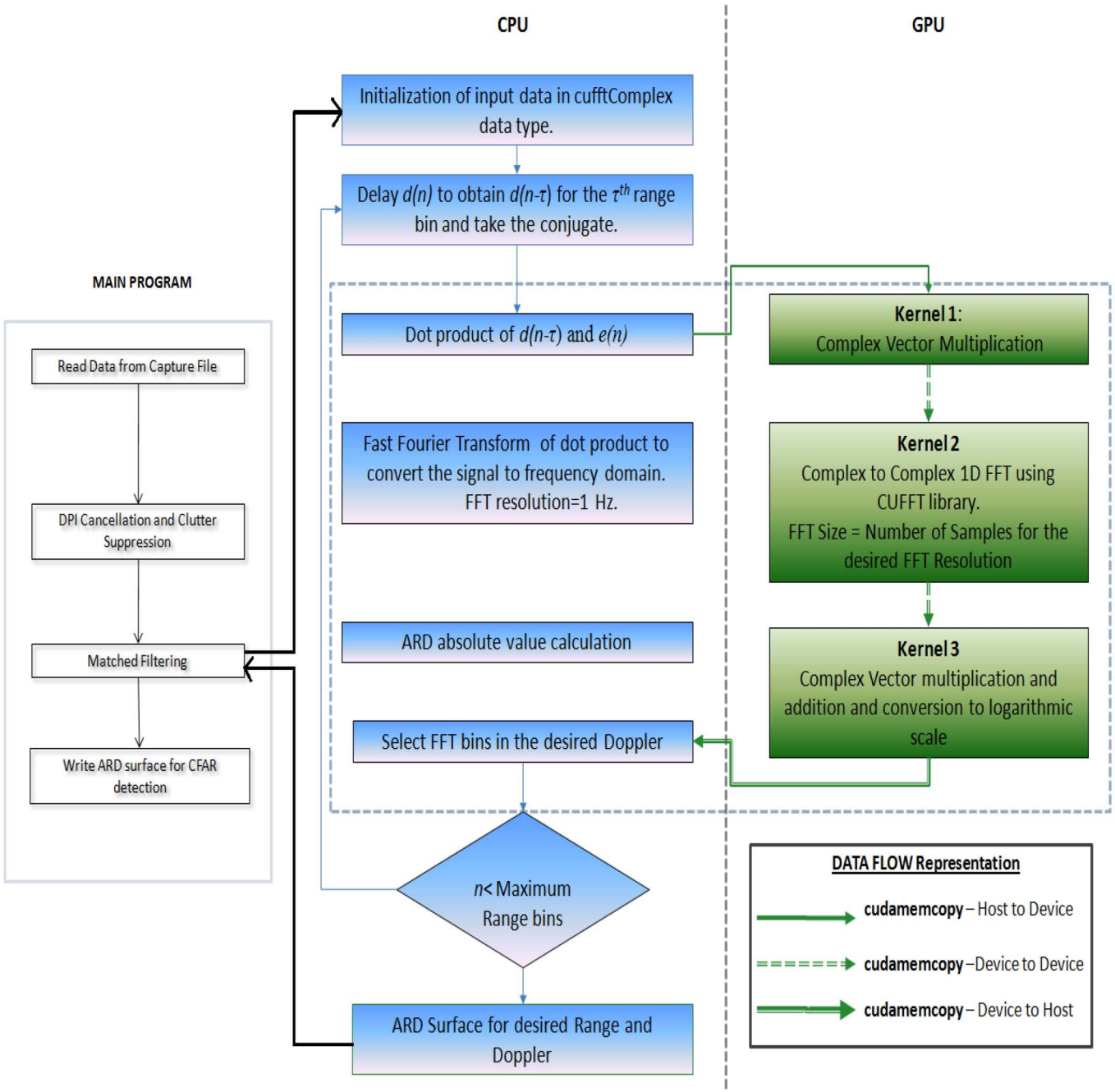

Figure 3.1: Flowchart of Matched Filtering/ARD Processing in GPU-CPU Heterogeneous Platform (The inset illustrates data flow directions)

- $A(n)$is the complex amplitude of direct signal received on surveillance channel
- c and τ_i are the complex amplitude and the delay (with respect to the direct signal) of i-th stationary scatter (i=1,...,N_c)
- a_k and τ_k are the complex amplitude and the delay of k-th target with f_{dk} Doppler frequency
- $v_R(n)$is the thermal noise contribution at the receiver antenna

The zero Doppler components in the signal $s_R(n)$ have to be removed for clutter and direct path interference removal.

$$s_{MP}(t) = \sum_{i=1}^{N_c} c_i s_T(t-\tau_i) + A(t) s_T(t) = \sum_{i=0}^{N_c} c_i s_T(t-\tau_i) \tag{3.3}$$

[25]

The signal $s_{MP}(t)$ is a linear combination of delayed replicas of $s_T(t)$ and complex coefficients c_i.

The operation of a linear adaptive filtering algorithm involves two basic processes :

1. A filtering process designed to produce a desired output in response to a noisy input.
2. An adaptive process that provides a mechanism for adaptive control of parameters, used in the filtering process.

Various adaptive filters were tested by the prior research [26] in this field, and NLMS filter was selected as the most optimum adaptive filter for the purpose.

3.3 Modelling of NLMS Algorithm

The NLMS algorithm is modelled from LMS filter [25], by normalising the coefficient update equation with respect to the squared-norm of the input data. The structure of NLMS defined by Haykin [25] in Figure 3.2 is used for algorithm modelling.

From Figure 3.2, the algorithm can be modelled as follows:

1. The user specifies the number of consecutive range bins, which refers to the maximum distance from the receiver up to which DPI and clutter cancellation have to be performed. The user also specifies the filter coefficient value μ_{NLMS} in Figure 3.2 for the adaptive control mechanism.
2. The algorithm begins with calculation of the zero Doppler/multiparty component in $s_R(n)$ mentioned in equation 3.2. The multipath component is named as $\hat{d}(n)$ with reference to figure.

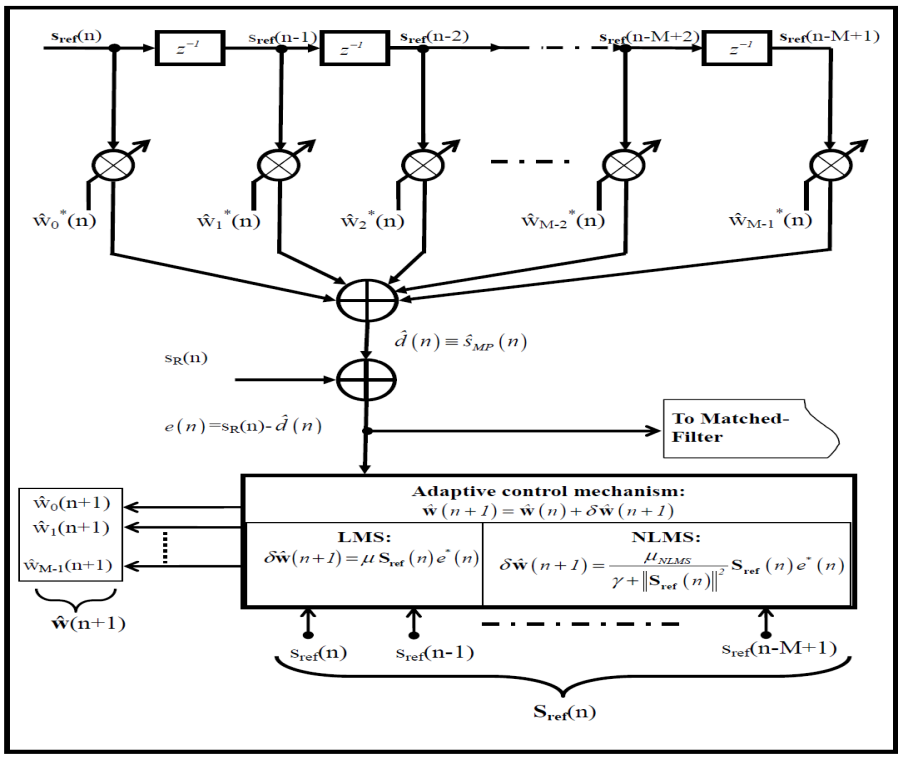

Figure 3.2: Structure of NLMS Filter [25]

3. $\hat{d}(n)$ calculated in step 3 is used to calculate the estimation error $e(n) = s_R(n) - \hat{d}(n)$ which acts as the input to the matched filter. At this stage the filtering process is complete and the algorithm proceeds to the adaptive control mechanism.

4. The adaptive mechanism is designed with reference to the tap-weight updation equation given by

$$\hat{w}(n+1) = \hat{w}(n) + \delta\hat{w}(n+1) \tag{3.4}$$

[25] where the correction factor $\delta\hat{w}(n)$ is given by:

$$\hat{w}(n+1) = \frac{\mu_{NLMS}}{\gamma + \left\|S_{ref}(n)\right\|^2} S_{ref}(n)\, e(n) \tag{3.5}$$

[25]

Inspection of the algorithm above reveals that the process consists of vector multiplication of delayed inputs, iterative summation etc. The NLMS algorithm was first designed in MATLAB for simulation purpose and was then converted to ANSI C [11].

3.4 CPU implementation of NLMS Algorithm

The CPU implementation is based on the algorithm model mentioned in Section 3.3. The variables and parameters are with reference to Figure 3.2.

1. Basic Steps

   (a) The number of filter taps/ number of range bins for multipath cancellation, and the $\mu_{NLMS}$is initialised with user-specified values in the command line.

   (b) The reference channel signal $S_{ref}(n)$ and the surveillance/echo channel signal $s_R(n)$ (equation 3.2)is initialised in float data type with separate variables for real and complex components.

2. Filter Output

   (a) The reference signal is delayed with respect to the number of filter taps mentioned in basic steps. In other words, the delay elements in Figure 3.2 are realised by simple array index manipulation.

   (b) Multipath component $\hat{d}(n)$ is calculated as the summation of delayed version of reference signal multiplied by the filter taps. This is achieved by multiplication and summation of separate I and Q components in iterative structure.

   (c) $\hat{d}(n)$ is subtracted from $s_R(n)$ to calculate estimation error $e(n)$.

3. Adaptive Control mechanism

   (a) Equation 3.5 mentioned in NLMS algorithm modelling, is realised in a single iterative loop consisting of calculation of correction factor $\delta\hat{w}(n)$, by separate vector multiplication and division, for I and Q and the tap-weight update equation 3.4.

   (b) The updated tap-weight $\hat{w}(n+1)$ is calculated by cumulative addition within an iterative structure.

3.5 GPU implementation of NLMS Algorithm

CPU implementation of NLMS algorithm reveals that the process consists of nested iterative structures. Before proceeding to GPU migration, timers for individual processes are set and the time-consumption is according to Table 3.2. GPU migration started aiming maximum speed-up for individual processes by identifying process according to criteria mentioned in Section 2.2.1. Unlooping of nested loop to batch process starts from the outermost loop, and proceeds to the innermost loop, implementing parallelism within parallelism [46]. The salient features of GPU implementation are as follows.

1. Basic Steps

   (a) The initialisation of filter-taps and μ_{NLMS} is done in the CPU and copied to the device memory.

(b) The reference channel signal $S_{ref}(n)$ and the surveillance/echo channel signal $s_R(n)$ (equation 3.2)is initialised as cuFloatComplex [48] data type instead of separate of separated I and Q data and is copied into device memory.

(c) The delaying of reference signal is done in the CPU due to the sequential nature of the process.

2. Filter Output

(a) Multipath component $\hat{d}(n)$ is calculated with a custom kernel written using cuBlas and algebraic operation for cuComplex [48] data type. The thread index [46] setting for this operation has to be set with reference to the number of delayed input samples from step 1 (c).

(b) Estimation error is calculated in a separate kernel, and the variables are transferred by a device to device transfer. The reason for separate kernel is the cumulative addition in Step 2(a), resulting in assigning of different thread Index.

3. Adaptive Control mechanism

(a) All the variables for adaptive mechanism are already present in the GPU and rest of the variables for calculating equation are initialised as device variables.

(b) $\delta\hat{w}(n)$ is calculated using custom kernel. cuBlas is used for multiplication, division and absolute value calculation. Nested cuBlas functions are implemented for this purpose. The nested parallelism at this stage is handled by the device itself and this acts as the supporting reason for the exclusive use of cuBlas for $\delta\hat{w}(n)$ calculation.

(c) The updated tap-weight is calculated from $\delta\hat{w}(n)$ within the same kernel mentioned in step 3(a) since the same thread index can be used.

Process	CPU Timing (sec)	GPU Timing (sec)	Speed-up Factor
Filter Output: Multipath Component	4.27	0.23	18.56X
Filter Output: Estimation Error	0.12	0.05	2.4X
Adaptive Control Mechanism	7.73	1.07	7.22X
Total Time	12.12	1.35	8.98X

Table 3.2: Comparison of CPU Vs GPU time consumption of individual processes in NLMS filtering for Data-Size= 819.2 K Samples, Order=300 and $\mu_{NLMS} = 0.02$.

3.6 Performance Optimisation of GPU Implementation

The performance optimisation of the GPU implementation is based on three basic strategies:

- Maximise parallel execution to achieve maximum utilisation
- Optimise memory usage to achieve maximum memory throughput
- Optimise instruction usage to achieve maximum instruction throughput.

The usage of these strategies depends on the program flow and the optimisation strategy matching for that particular portion of the program should be implemented. The overall performance optimisation is hence achieved by applying the suitable optimisations for each stage.

3.6.1 Program Flow

The Program flow consisting of both DPI and Clutter Cancellation, and matched filtering is explained here. The Program can be divided into four phases-Reading Capture File, DPI and Clutter Cancellation, Matched Filtering and Writing ARD values to file. The cycle repeats for each second of observation, or in other words for each data set.

1. Reading Capture File in Main Program (Figure 3.3)
   (a) The capture file written in ".rcf " [55] format by the acquisition stage is read into separate reference and surveillance signal in cuFloatComplex data type [48].
   (b) The data size at each read is equal to the number of samples expected for each ARD surface as specified by the user.
2. DPI and Clutter Cancellation
   (a) The echo and surveillance data is passed to NLMS CPU subroutine.
   (b) The control and data is passed onto GPU as illustrated in Figure 3.3.
   (c) The Clutter cancelled data is returned to the main program.
3. Matched Filtering
   (a) The Clutter cancelled data is passed from main program together with the desired range and Doppler [57].
   (b) The control and data is passed onto GPU as illustrated in Figure 3.1.
   (c) Both the absolute value and the linear value of ARD surface is returned to main program.
4. Writing ARD to file
   (a) Each ARD value calculated is written into ".ard" [55] file.
   (b) The ARD value can be written either in linear or logarithmic scale as specified by the user.

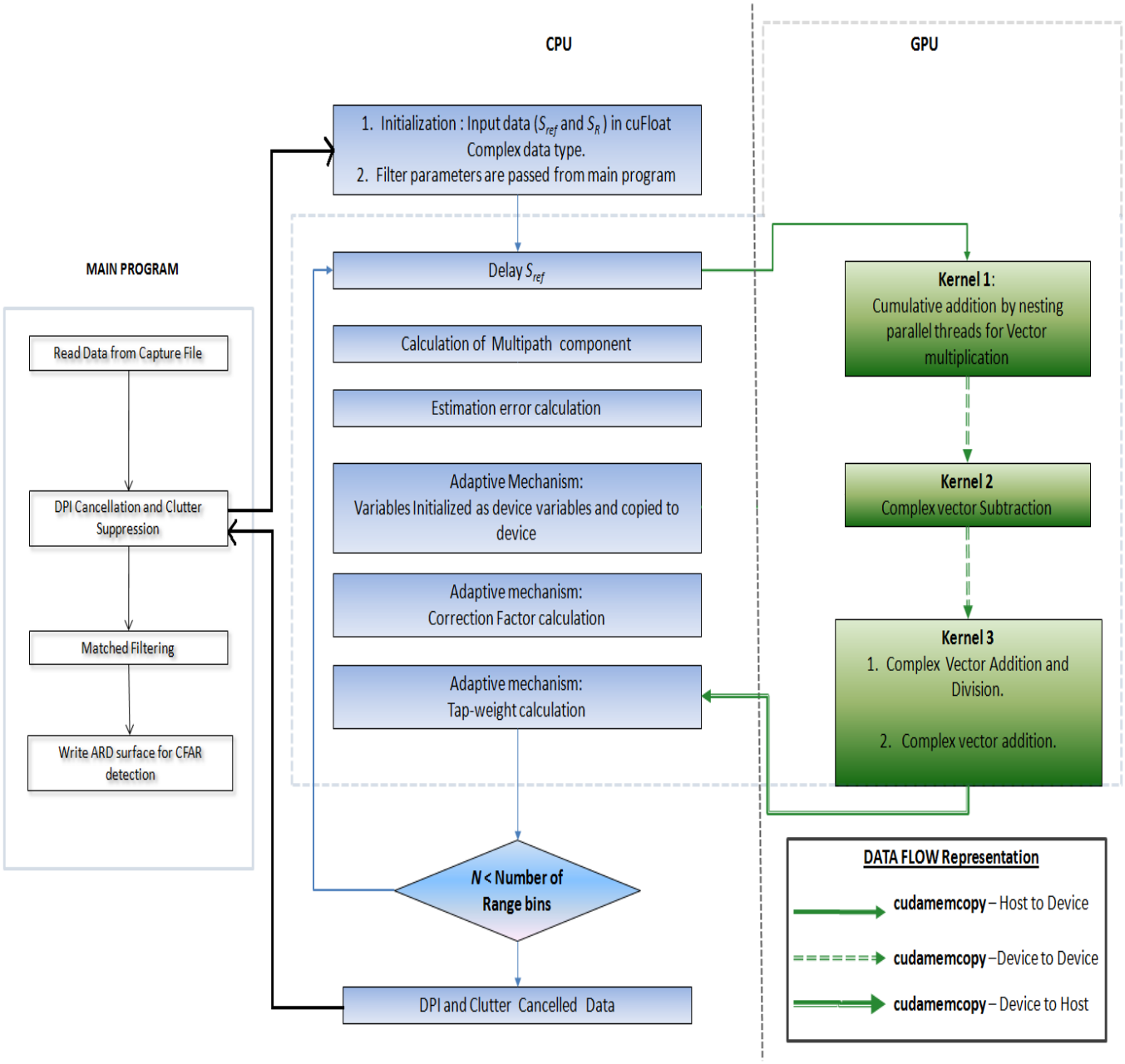

Figure 3.3: Flowchart of NLMS clutter cancellation in CPU-GPU heterogeneous platform (The inset illustrates data flow directions)

3.6.2 Optimisation Measures

- To minimise data transfer between the host and the device, the output at intermediate stages is retained in the device and is used by the next process. Only the control variables for that particular process is transferred from the host. Hence device to device transfer is exploited to the maximum.
- Sequential process like loops with comparatively less iterations are performed in the CPU. This includes delaying reference signal in ARD processing.
- CUFFT can be used to calculate FFT for up to 8 million data samples [44] in a single instance. But the intention of program is to track the target at each second which corresponds to nearly 409.6K Samples/ ARD surface. This data rate is used for the test deployment using FM illumination, which will be discussed in Chapter 5. But tests using DVB-T data was at a sample rate of 8 million samples/ARD surface, utilising the maximum throughput of the device.
- Instruction throughput is achieved by utilising the same kernel for multiple vector operations. But most of the kernels are custom written for that particular operation.

3.7 Pros and Cons of NLMS algorithm

The Pros and cons of NLMS algorithm is discussed with respect to two criteria- The Computation efficiency and the Clutter Cancellation efficiency. Detailed algorithm comparison will be done in Chapter 5, though this Section provides an overview of NLMS algorithm results.

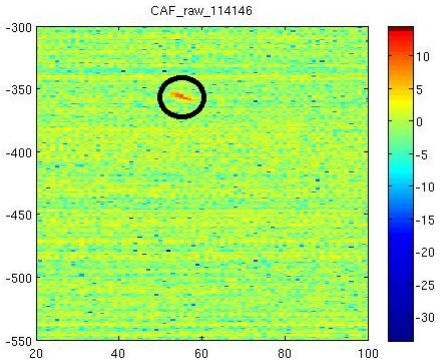

Figure 3.4: Target detection (circled) at (55,-350) in CPU using DVB-T data with NLMS Algorithm

3.7.1 Pros of NLMS Algorithm

- NLMS Algorithm gave very good result with DVB-T as the source of illumination. The data was obtained from Pisa, Italy and has a sampling frequency of 8.5 MHz. Identical ARDs with

respect to target location in range and Doppler was obtained in both CPU and GPU. Figure 3.4 and Figure 3.5 shows ARD obtained for DVB-T data in CPU and GPU respectively.

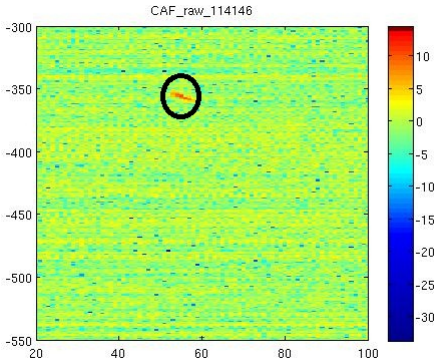

Figure 3.5: Target detection (circled) at (55,-350) in GPU using DVB-T data with NLMS Algorithm

- Table 3.2 reveals that NLMS is a fast processing algorithm with respect to GPU computation. Detailed study on the processing time will be discussed in Chapter 5.

3.7.2 Cons of NLMS Algorithm

Though NLMS algorithm is a fast processing algorithm with good results, the algorithm is not suited for PMR due to the following reasons:

- NLMS Algorithm is depended on Source of illumination
  - μ_{NLMS} have to be set for each source of illumination. Optimum value of μ_{NLMS} is calculated by trial and error method, which is not practical in a networked system with multiple transmitters, and will affect the reliability and robustness of the whole system.
- Inefficient in Strong Clutter Environment

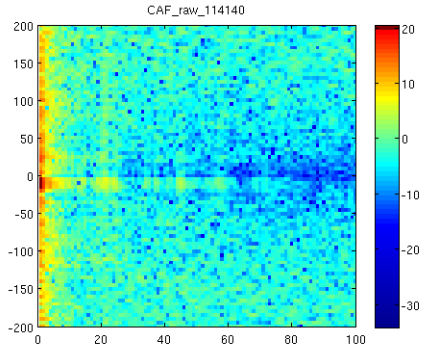

Figure 3.6: Target Masked by Clutter when using NLMS algorithm

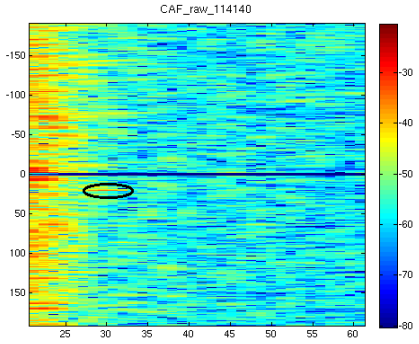

Figure 3.7: Target detected (circled) at (30,25) when using ECA algorithm

– The algorithm was not able to detect targets in strong clutter environment. The ARDs shown in Figure 3.6 and Figure 3.7 illustrates this point. The target was visible when using ECA. Detailed comparison of ECA and NLMS with respect to Signal to Clutter(SCR) ratio will be discussed in Chapter 5.

3.8 Conclusion

This Chapter basically explained the GPU implementation details of Matched Filtering [57] and NLMS algorithm [25]. The Chapter discussed the algorithm modelling of Matched filtering. The GPU implementation of matched filtering was explained in detail, with reference to flowchart featuring data flow directions, between host and the device in heterogeneous platform. The Chapter in its second half, explained NLMS algorithm which was the first algorithm selected for DPI and clutter cancellation [57]. The modelling of the algorithm from filtering model was discussed. The GPU implementation of the modelled algorithm was discussed with reference to flowchart. The processing time of individual processes in both the platforms were compared for both matched filtering and NLMS algorithm. The performance optimisation measures implemented, were discussed in detail with explanation of the program flow. Finally, the Chapter depicted pros and cons of NLMS algorithm. Though NLMS algorithm is computationally efficient, the cons of the algorithm discussed, necessities an alternate clutter cancellation algorithm which is robust and will increase the overall reliability of the system. Chapter 4 begins with the choice of ECA as the alternative to NLMS and proceeds to the implementation details of ECA.

Chapter 4

Extensive Cancellation Algorithm

The PMR project at RRSG aims to develop a radar technology that can compete at some level of performance with more traditional air traffic control radar [29]. The NLMS algorithm [25] modelled and implemented in Chapter 3 is a computationally efficient algorithm, but the NLMS filter has to be adapted to the signal used with the input variable μ_{NLMS} [25]. The ideal PMR system is expected to deliver reliable performance in different signal environments. The need for a signal independent DPI and clutter cancellation algorithm arises at this point. Such an algorithm is inevitable with reference to the cons of NLMS discussed in subsection 3.7.2. The ECA [25] is modelled and implemented as a solution to improve the reliability and robustness of the PMR system.

This Chapter begins with a very brief review of ECA and on the earlier work using this algorithm. This includes an introduction to the algorithm modelling and the MATLAB implementation [26] of ECA. The Chapter then proceeds to the implementation of ECA in C++ language using Armadillo [52] library. The complexity of the algorithm is explained, with reference to the computational requirement expressed in the total number of floating point operations and memory bandwidth required, followed by the supporting factors for ECA implementation in the GPU platform. The core of this Chapter is the GPU implementation of ECA, with reference to flowcharts and program flow. The Chapter concludes with the implementation details of GPU based high performance complex matrix inversion, which can be used for other DSP-based [10] applications.

4.1 Theory of ECA

NLMS algorithm discussed in Section 3.3 is a least mean square [53] approach whereas, ECA is a least square [53, 25] approach. The algorithm is discussed in detail in [15, 25]. The correction factor, $e(i)$ mentioned in Figure 3.2, can be rewritten for ECA in data matrix form [25] as:

$$e = s_R - X\left(X^H X\right)^{-1} X^H s_R = \left(I - X\left(X^H X\right)^{-1} X^H\right) s_R = P_0 s_R \tag{4.1}$$

In equation 4.1, X refers to the clutter subspace matrix [25] and the projection operator $P_0 = \left(I - X\left(X^H X\right)^{-1} X^H\right) s_R$, projects the received vector s_R in the subspace orthogonal to the clutter subspace. e represents the received signal with the zero-Doppler component cancelled. The clutter cancellation can be extended beyond zero-Doppler by extending the dimension of X by including Doppler-shifted replicas of the reference signal. With the addition of this process, the algorithm is called Extensive Cancellation Algorithm or ECA. The multiple matrix product $\left(X\left(X^H X\right)^{-1} X^H\right)$ in equation 4.1 leads to the computational complexity of the algorithm, with the increase in number of data samples and the number of range bins. ECA is a batch processing approach and hence divides the input signal into equal blocks for clutter cancellation.

4.1.1 Modelling of ECA

1. ECA algorithm begins with the creation of reference matrix X mentioned in equation 4.1. X is the Doppler [54] shifted replica of reference signal S_{ref} in the clutter subspace with dimensions depending upon the number of range bins selected for DPI/multipath cancellation[57].
2. The next step is the realisation of $\left(X\left(X^H X\right)^{-1} X^H s_R\right)$. The multiple matrix product consists of the following complex matrix operations:
   (a) Complex matrix multiplication.
   (b) Hermitian transpose.
   (c) Complex matrix inversion.
3. The final step is calculating $e = s_R - \left(X\left(X^H X\right)^{-1} X^H s_R\right)$, which acts as the input for ARD processing discussed in Chapter 3.
4. The algorithm also features the translation of the cancellation zone in range and Doppler axis in the ARD plot, which enables selective cancellation of specific Doppler and range. ARD plots supporting selective cancellation feature of ECA will be discussed in Chapter 5.

4.2 MATLAB Simulation of ECA

The MATLAB implementation is mentioned in detail in the earlier work [26] on ECA. MATLAB as a simulation platform performed the process straightforward. The salient features are mentioned here:

1. Realisation of X is done using iterative structures.
2. The estimation error, $e = s_R - \left(X\left(X^H X\right)^{-1} X^H s_R\right)$ is implemented using standard MATLAB codes.

3. Due to the computational complexity, the process is extremely time consuming and hence the data was divided into batches of 250K complex samples. Matched filtering for this relatively smaller data size reduces the integration gain [57] and is not preferred for efficient tracking using CFAR [41] and because of this, the ARD processing could be done only once clutter cancellation is completed for several batches of 250K complex samples .

4. The PMR system is expected to have a range coverage of 100-200 km and hence the cancellation has to be performed for a minimum of 300 range bins depending upon range resolution [57]. This tremendously increases the processing time.

5. MATLAB code can be used only for simulation purposes and cannot be used as an embedded code in commercial systems. However, the MATLAB implementation was inevitable in the preliminary testing of ECA.

4.3 Implementation of ECA in C++

The need for embedded processing led to the development of code in C++. But the algorithm complexity and the complex matrix inversion were not favourable for the usage of standard C/C++ routines. Armadillo [52] library which is a C++ matrix algebra library was used at this stage. The library is used together with BLAS [7] and ATLAS [6] library. The main features of C++ implementation are:

1. Variables representing S_{Ref} and s_r are initialised in colvec matrix data type [52] with separate real and imaginary parts.

2. Clutter subspace matrix is created using the linespace function [52] for time delaying in range.

3. Estimation error is implemented with a similar code structure as in MATLAB implementation. The analogy of Armadillo [52] with MATLAB is evident at this stage.

4. Armadillo [52] based C++ implementation is successful for embedded processing, but the processing time increased tremendously compared to MATLAB.

4.4 Computational requirement of ECA

It is important to analyse the computational requirement of ECA before proceeding to the implementation on the GPU. The number of floating point operations in ECA is defined by the equation [25],

$$(NM^2 + M^2 \log M)$$

where,

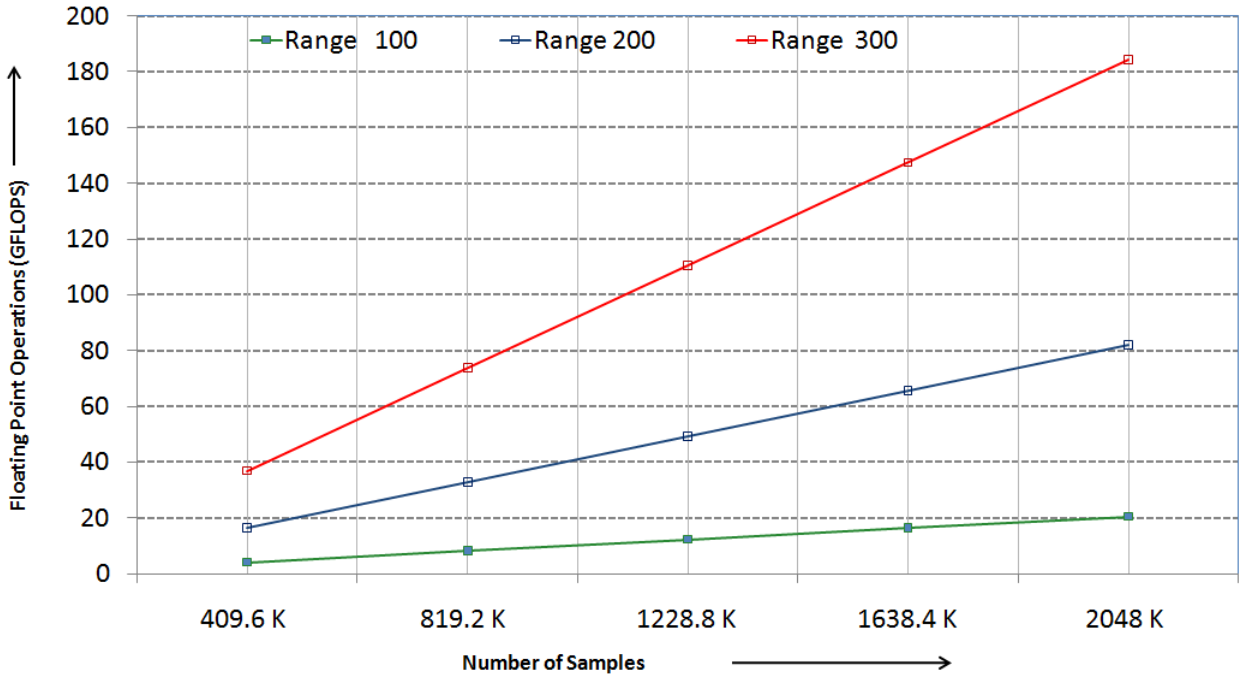

Figure 4.1: Number of floating point operations in ECA for Doppler bins=1 for various range and size of input data.

M=(Number of Range bins X Number of Doppler bins) and N= Number of Data Samples

Figure 4.1 shows the number of floating point operations in ECA in GFLOPS with varying value of range bins and for different data sizes. The number of Doppler bins is set to 1.

Figure 4.1 reveals that the computational requirement increases tremendously with increase in the number of range bins. When the processing is done for 2048 K samples and for 300 range bins, the computational power required is 184.3 GFLOPS. This is the computational requirement for ECA alone.

ARD processing has one FFT operation consisting of $5N\log_2 N$ operations added with two N point multiplications leading to a total of $5N\log_2 N + 2N$ floating point operations, where N is the number of data samples. The addition of this value will not make a significant increase to the overall computation requirement. From Figure 4.1 it is evident that the real time processing of PMR system with ECA clutter cancellation cannot be handled with an AMD Phenom II X4 955 Processor which is the CPU platform used and has a theoretical peak performance of 51.2 GFLOPS (Refer Appendix A) even if all the cores are utilised. GTX 480 FX on the other hand has a theoretical performance of 1848 GFLOPS [8] which is sufficient for real time processing of PMR signal processing. But the available memory bandwidth is a major factor that affects the effective performance. The process requires a memory bandwidth of 184.3 X 2 =368.6 GB/s. The available theoretical bandwidth [49] of GTX 480FX is 177.4 GB/s [8]. The effective bandwidth [49] in general will be less than the theoretical bandwidth [49]. Detailed study on the memory bandwidth perfromance is given in the Section 4.5.

4.5 Memory bandwidth performance

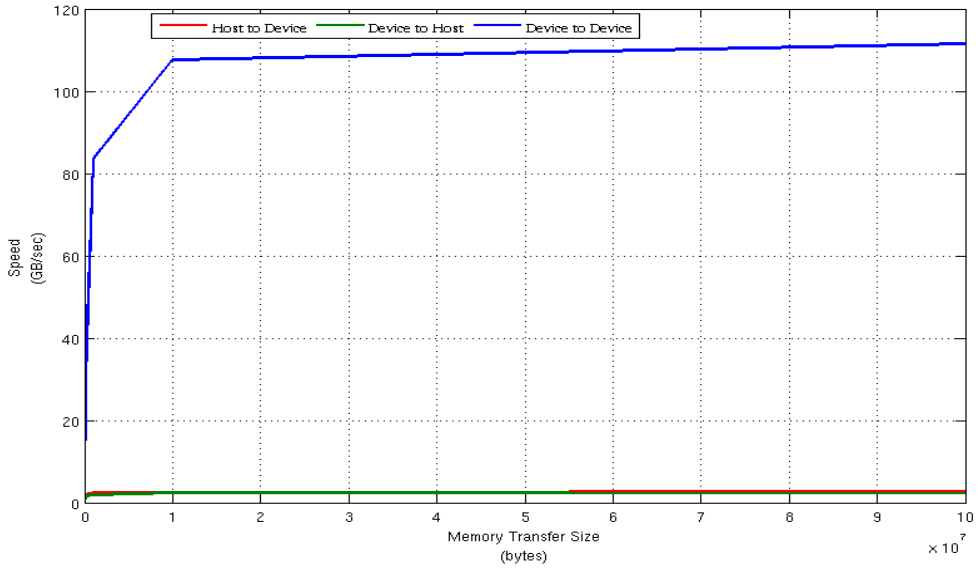

Figure 4.2: Effective Bandwidth variation with data size

4.5.1 Theoretical Bandwidth

The theoretical bandwidth of GPU is calculated from hardware specifications (refer Appendix B). Using this data, the theoretical bandwidth of Nvidia GTX 480 FX is calculated as [49]:

$$(1848\,X\,10^6\,X\,(384/8)\,X\,2)/10^9 = 177.4\,\text{GB/sec}$$

where $1848\,X\,10^6$ is the memory clock in Hz, 384 is the memory interface width converted to bytes on division by 8. The multiplication by 2 is due to the double data rate, and is finally converted to GB/sec on division by 10^9.

4.5.2 Effective Bandwidth

The graph in Figure 4.2 is plotted from the output of *banwidthTest* program in CUDA SDK. It is observed that the peak bandwidth for device to device Transfer at a data transfer rate of 10^8 bytes is 111.51 GB/sec. As expected, the effective bandwidth is much lesser than the theoretical bandwidth. Hence the available memory bandwidth of a single GPU is not sufficient for real time processing of PMR system with reference to Section 4.4. Hence real time processing of PMR system is expected only in a multi-GPU environment by splitting the data between the devices.

4.6 GPU implementation of ECA

4.6.1 Supporting factors for GPU implementation

- The characteristics of ECA as a batch process fits ideally into the criteria for parallel processing mentioned in Section 2.2.1
- The algorithm model reveal the multiple matrix product as the major phase of processing. The phase can be implemented using cuBlas and extending the CUDA Software Development kit (SDK) [46].
- A major portion of the time consumption and algorithm complexity is due to the hermitian transpose and complex matrix inversion of the large matrix. GPU processing upon speeding up this process will significantly add the overall speed-up factor.
- The data type used for CPU implementation considered separate real and complex parts for the surveillance and reference signal. In CUDA, all the variables are declared in cuDoubleComplex [48] data type. Thus the overhead to the processor due to type conversions is eliminated.
- The effective speed-up achieved with single-GPU is not expected to be sufficient for real time processing with reference to analysis of available memory bandwidth from Section 4.4. But, the batch process nature of ECA and the scalability of the code helps in easy migration to a multi-GPU platform, and hence real-time processing can be achieved.
- The data acquisition code and the steps that follow ECA and matched filtering including CFAR detection [41] are written in C++. The analogous nature and compatibility of CUDA with C/C++ favours the use of GPU platform with the CUDA toolkit.

4.6.2 Implementation

The GPU implementation starts with the identification of processes, that can be processed using parallel GPU threads. Algebraic operations using large data size are selected for GPU migration. Based on this strategy, ECA can be divided into three phases.

Features of GPU migration within each phase is discussed here. GPU based complex matrix inversion achieved major performance speed-up and is discussed as a separate Section for use in other DSP [10] algorithms.

1. Building of clutter subspace matrix X
   (a) The calculation of time intervals from the sampling frequency (409.6kBs used for data capture). The process is done using a custom kernel to perform the division on real data.
   (b) The calculation of elements in matrix X is done using complex vector multiplication.

2. Calculation of Multiple matrix product $\left(X\left(X^HX\right)^{-1}X^Hs_R\right)$

   (a) X^H is calculated as the hermitian transpose of the complex matrix X. Matrix X has N rows and M columns, where N= Number of Data Samples and M=(Number of Range bins X Number of Doppler bins) , and hence X^H is of dimension (M,N) in row major format. Hermitian transpose is accomplished by customising the real matrix transpose code from CUDA SDK with a subroutine for complex matrix transpose.

   (b) $\left(X^HX\right)$ is performed using shared matrix multiplication extended for complex matrix multiplication giving a square matrix of order M .

   (c) $\left(X^HX\right)^{-1}$is the most time consuming portion of processing and is done fully in GPU. Section 4.7 discusses complex matrix inversion in detail.

   (d) $X\left(X^HX\right)^{-1}X^Hs_R$ is realised with the same matrix multiplication kernel designed for use in step (b).

3. Estimation error e is calculated by kernel designed for complex vector subtraction.

4.6.3 Program Flow

The Program flow of ECA illustrated in Figure 4.3. As mentioned before, the characteristic of ECA as a batch process is exploited to the maximum extent in the implementation.

- The main program passes input data consisting of reference and surveillance signal to the CPU subroutine for DPI and clutter cancellation .
- Processes, from clutter subspace matrix creation to estimation error calculation take place in the GPU, and the estimation error, which is a column matrix of dimension equal to number of data samples is copied back to the CPU. The program flow thus reduces memory transfer between host and device to the minimum.
- The data transfer at this stage is completely device to device and the same kernel is used multiple times for processes like matrix multiplication, as it is evident from Figure 4.3 .
- The batches are executed in parallel at multi-processor level. The processes within each batch are executed within a single streaming processor. This implementation plan exploits maximum usage of all the cores of the GPU as mentioned in subsection 3.6.

4.7 GPU based Complex Matrix Inversion

Complex matrix inversion is an inevitable part of many DSP algorithms. The scope of a high performance complex matrix inversion is therefore not limited to PMR Signal Processing. The complex matrix inversion mentioned in Figure 4.3 is explained in this section.

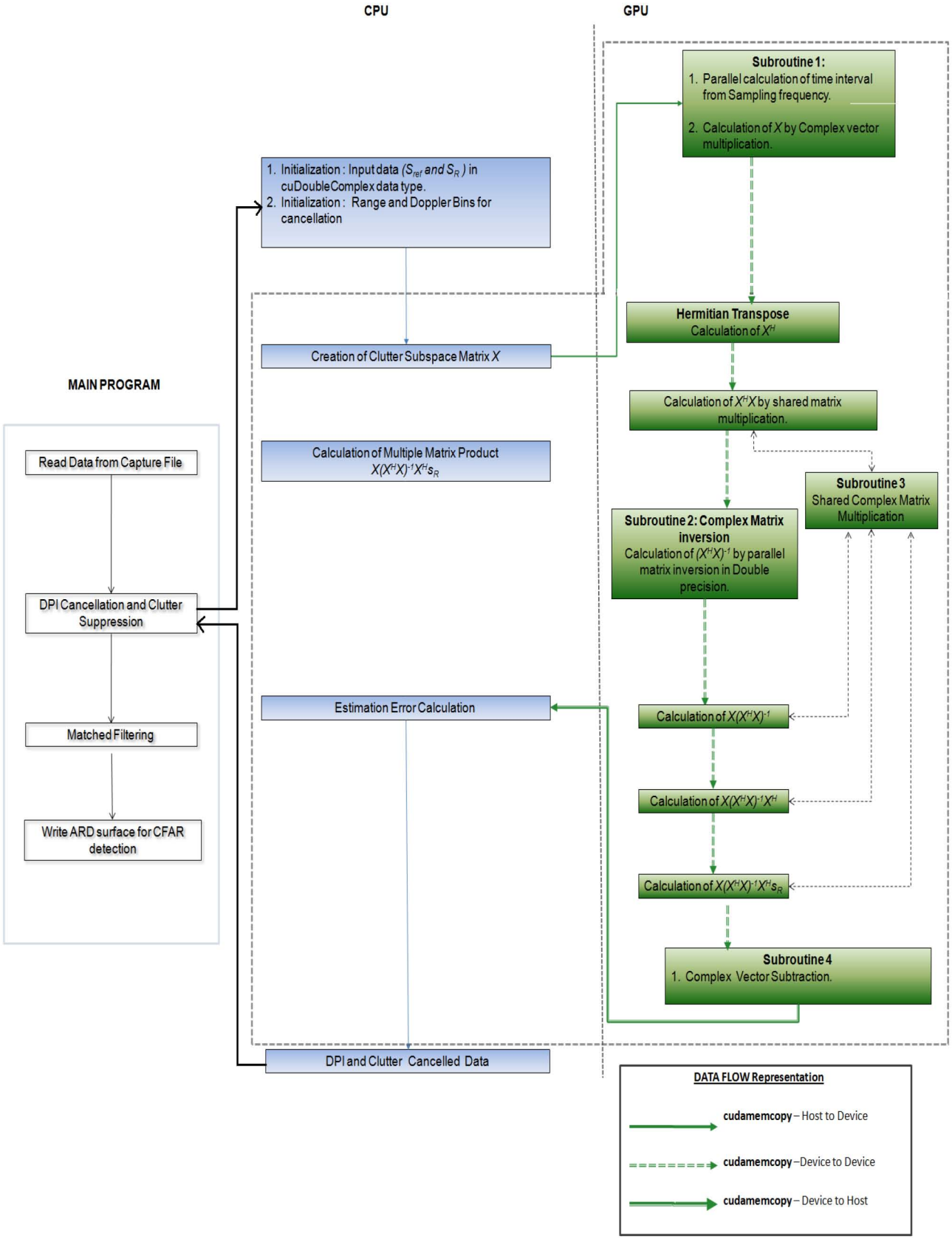

Figure 4.3: Flowchart of ECA clutter cancellation in CPU-GPU heterogeneous platform (The inset illustrates data flow directions)

4.7.1 Implementation Features

The matrix inversion is done by Gaussian elimination [3]. The program is adapted from real matrix inversion. The real matrix inversion code was converted to complex matrix inversion by changing the variables to cuFloatComplex data type [48] with appropriate changes in the intermediate stages. The preliminary code developed with this criteria had the limitation of input matrix size up to 400. This problem was solved by an algorithm mentioned in [20]. This algorithm uses real matrix inversion to calculate complex matrix inversion. The algorithm is mentioned in Algorithm 4.1. All the intermediate multiplications in the algorithm are performed in GPU, which further increased the performance of the program.

Algorithm 4.1 Complex matrix inversion from real matrix inversion [20]

- Input: Complex Square Matrix- $(A+iC)$, Order $N\ X\ N$
- Initialisation: The Input matrix is initialised as two real matrices A and C.
- Computation:

  $r_0 = A^{-1}.C$

  $y_{11} = (C.r_0 + A)^{-1}$

  $y_{01} = r_0.(y_{11})$

  $y_{10} = -y_{01}$

  $y_{00} = y_{11}$

- Output: Inverted matrix- $(y_{00} + iy_{10})$

4.7.2 Program Flow

Salient features of the program flow illustrated in Figure4.4 are:

- Inspection of Algorithm 4.1 and flowchart in Figure 4.4 reveals that two real matrix inversions are taking place in the implementation. They are done in double precision in ECA for better results.
- The intermediate stages in the algorithm are calculated within the GPU by “device to device” memory transfer to the two separate kernels for shared matrix multiplication and matrix addition.
- The number of allocated threads can be increased depending on the device, when the code works as a standalone program.

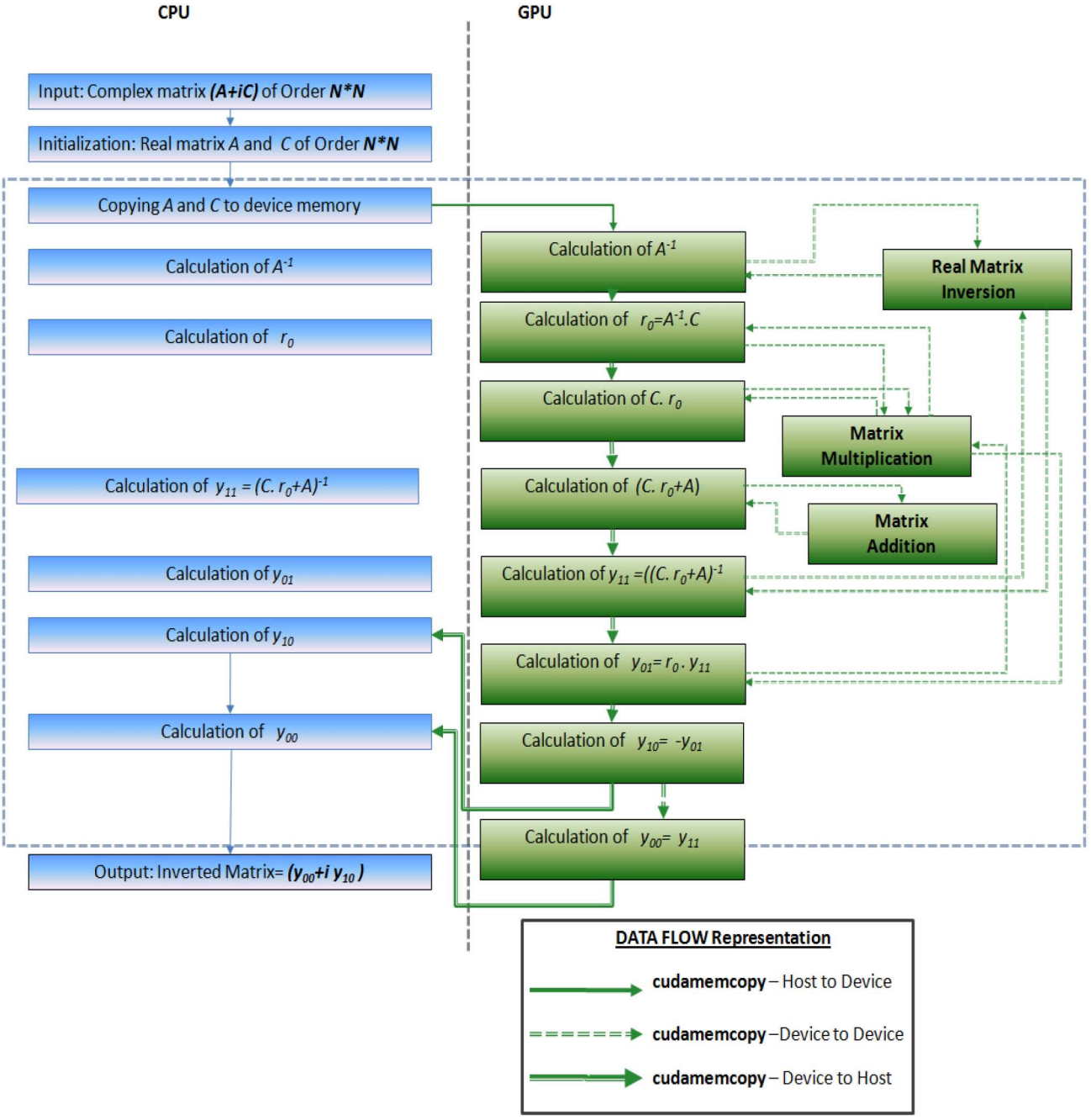

Figure 4.4: Flowchart of GPU Complex Matrix Inversion (The inset illustrates data flow directions)

- The complex matrix inversion block is customised for ECA, with final output transferred to next step mentioned in Figure 4.3 by device to device transfer.

4.7.3 Factors affecting Speed-up

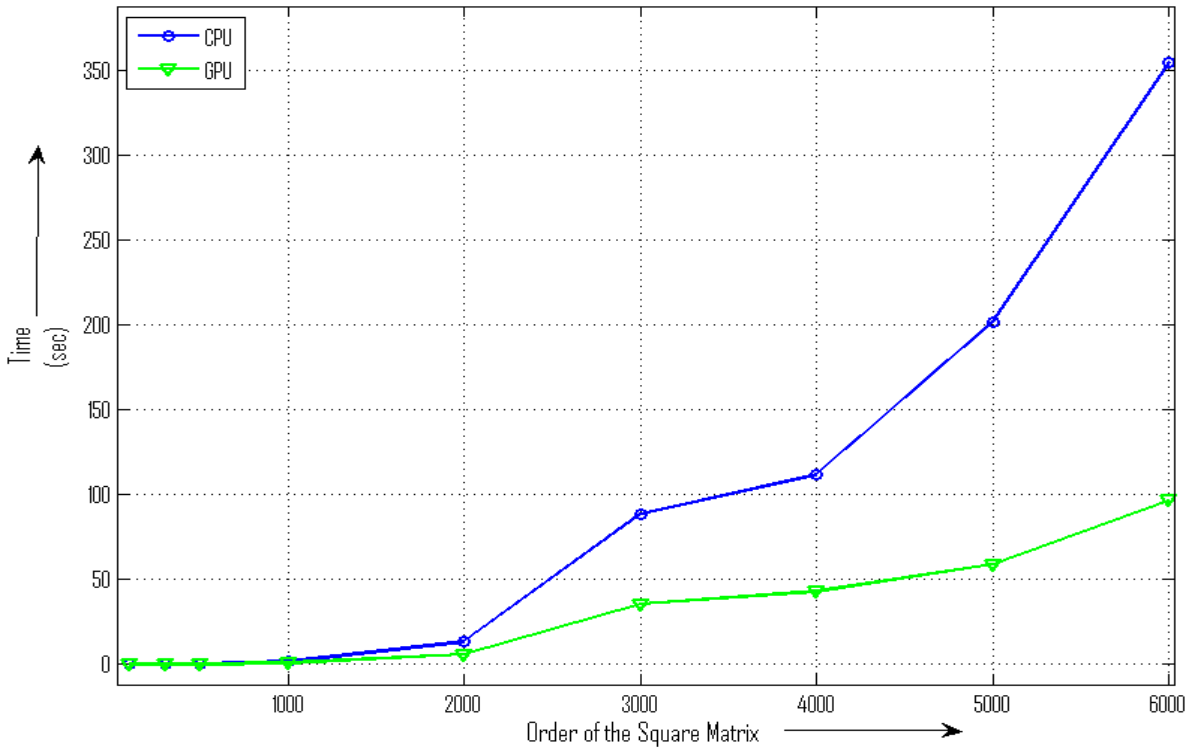

Figure 4.5: Processing Time for complex matrix inversion: CPU vs GPU

Figure 4.5 compares the processing time for complex matrix inversion in CPU and GPU. Figure 4.6 shows the variation of speed-up factor with increasing matrix order. From Figure 4.6 and Figure 4.5, it is observed that up to a matrix size of 250 X 250, the CPU implementation is better than GPU implementation since the speed-up factor is less than or equal to 1. When the size of the matrix increases, the speed-up factor increases considerably up to value of 3.5X. This is due to the number of allocated parallel threads. For small matrix sizes, the number of parallel threads allocated is less and the flow-control data structures and memory transfer between the host and the device results in higher total processing time. However when the matrix size increases, the speed-up achieved by execution of more parallel threads compensates for the memory transfer time and results in an enhanced speed-up factor.

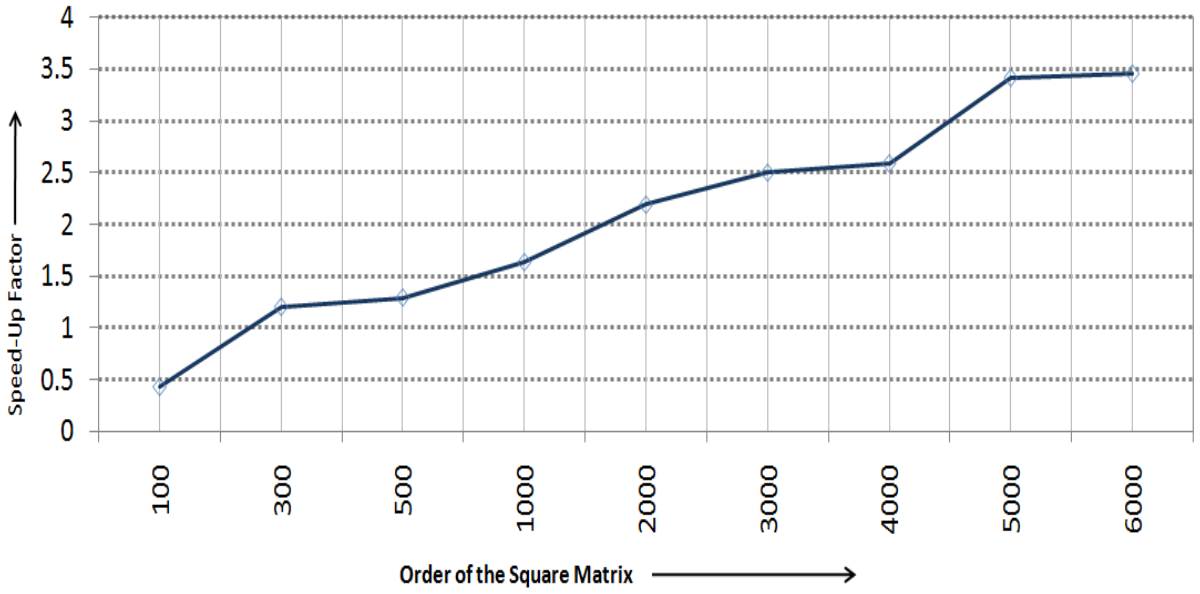

Figure 4.6: Speed up factor variation with order of the input square matrix

4.8 Conclusion

This Chapter introduced ECA as an alternative to the NLMS algorithm for DPI and clutter cancellation. The computational requirement of ECA provided evidence that GPU is a promising platform for accelerating ECA provided that the memory bandwidth is used efficiently. Salient features of MATLAB and C++ implementation of ECA were studied before proceeding to the GPU implementation. The GPU implementation of ECA was discussed in detail with reference to flowcharts and program flow. GPU based complex matrix inversion implementation as a part of ECA was also discussed as a separate section with a study on the factors affecting the performance. Detailed tests and results based on GPU implementation of ECA will be discussed in Chapter 5.

Chapter 5

Tests and Results

In Chapter 3 and Chapter 4, we discussed the modelling and implementation details of matched filtering and the two clutter cancellation algorithms- the NLMS and the ECA. This Chapter illustrates the testing of these accelerated processing algorithms implemented in GPU platform. The Chapter begins by outlining the test methodology, stating the parameters and features of the testing environment. The tests and results can be broadly classified into the following two categories:

- Computational Characteristics-
  - The memory bandwidth performance is studied by comparing the theoretical bandwidth and the effective bandwidth achieved with variation of the memory transfer size. The results of this study was already discussed in Section 4.5.
  - The GPU implementation of ARD processing, NLMS algorithm and ECA are analysed with respect to their performance metrics. The actual processing time between CPU and GPU and the speed-up factor is analysed for various data sizes and input parameters.
  - This analysis is further utilised for comparing the two clutter cancellation algorithms based on their computational efficiency.
- Algorithm Efficiency
  - The two algorithms are compared in terms of their clutter cancellation and target detection ability.
  - Detailed SCR analysis is performed to compare the clutter cancellation efficiency.
  - The two algorithms are tested with simulated data for various target sizes and with different sources of illumination.

The Chapter also gives results from the test deployment of the system in real world scenario. Validation of the results with respect to CPU processing is done by comparing the ARD plots. The Chapter then proceeds to overall performance gain achieved by the PMR signal processing chain in GPU

platform. Based on the broad category of tests and results mentioned above, a conclusion about the most appropriate algorithm to use in PMR signal Processing in GPU platform is done at the end of this Chapter.

5.1 Test Methodology

The testing of the implementation is according to methodology recommended by Nvidia for performance metrics analysis [49]. The features and parameters of the test environment are as follows:

- Platforms
  - The CPU platform used is a single threaded AMD Phenom II X4 955 3.2 GHz Processor with 16GB RAM. All speed and other performance comparison with the GPU implementation is with the C ++ code running on this single threaded CPU platform. It is to be noted that the CPU implementation can be improved substantially by multi-threading. Detailed specification of the CPU platform is given in Appendix A .
  - The GPU platform is Nvidia GTX 480FX with CUDA SDK version 3.0 on the same computer. More device specification is included in Appendix B .
- Speed-up Factor
  - The Speed-up Factor is calculated as recommended by Nvidia as the ratio of CPU processing time to GPU processing time.
  - The timers used are CUDA GPU timers [49].
- Memory Bandwidth
  - The theoretical bandwidth is calculated from hardware specification.
  - The effective bandwidth is calculated by timing specific program activities. The effective bandwidth EB is calculated as recommended by Nvidia [49] as:

$$EB = ((B_r + B_w)/10^9)/t$$

    Here, EB is in units of Gbps, $B_r$is the number of bytes read per kernel, $B_w$is the number of bytes written per kernel, and time t is given in seconds.
  - For calculation of bandwidth Gbps, the division is done by $10^9$and not 1024^3.
- Processing Time
  - The real processing time of the individual processing stages and the total processing time for the complete signal processing chain in GPU platform is compared with the corresponding processing time taken by CPU.

5.2 GPU ARD computation

5.2.1 Factors affecting Performance of GPU-ARD

Matched Filtering which generate ARD plots showing target detections, was one of the major time-consuming processes in PMR signal Processing. GPU implementation of ARD described in Chapter 3, has achieved a major speed-up over the CPU implementation.

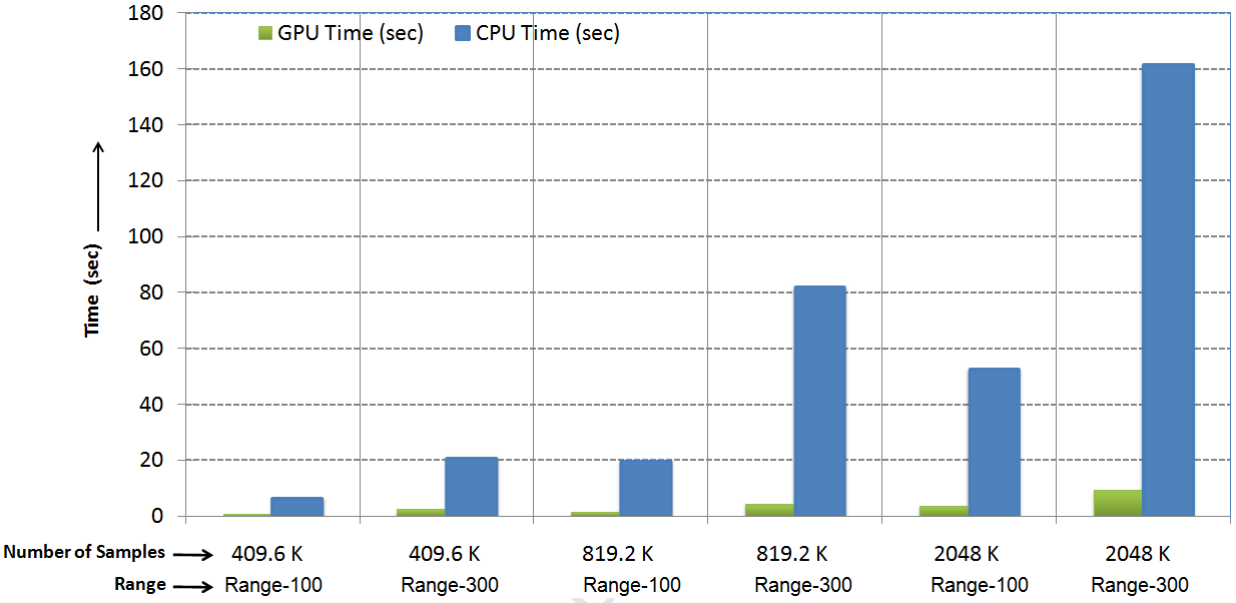

Figure 5.1: ARD Time Comparison-CPU vs GPU for 500 Doppler bins .

Figure 5.1 illustrates the CPU and GPU time consumption for ARD processing for different data sizes and for different number of range bins. Figure 5.2 illustrates the variation of speed-up factor. It is observed from Figure 5.2 that the speed-up factor increases with data size and number of range bins up to a maximum value of 18.67X. Memory bandwidth and the number of parallel processing threads are the key factors for the variation of speed-up. The effective data size increases with the increase in the number of range-bins for the dot product calculation and the FFT calculation illustrated in Figure 3.1. CUFFT library [44] for complex to complex one dimensional FFT has an upper limit of 8 million samples in a single execution. Therefore speed-up factor for FFT calculation will increase with the increase in data size up to 8 million samples, provided the data size is less than the available memory bandwidth. In the present scenario, the data size is below 8 million and hence the speed-up factor increases with the increase in size of data, due to more threads executing in parallel as observed in Figure 5.1 and Figure 5.2.

The speed-up factor variation for Range 300 exhibits a peak at a data size of 819.2K Samples and then decreases when the data size increases to 2048K Samples. The reason is the lack of memory bandwidth with the increase in the effective size of data. For range 200 and range 100, the speed-up factor further increases when the data size increase to 2048K Samples because the effective data size is below the total available memory bandwidth. This variation of speed-up factor is observed when

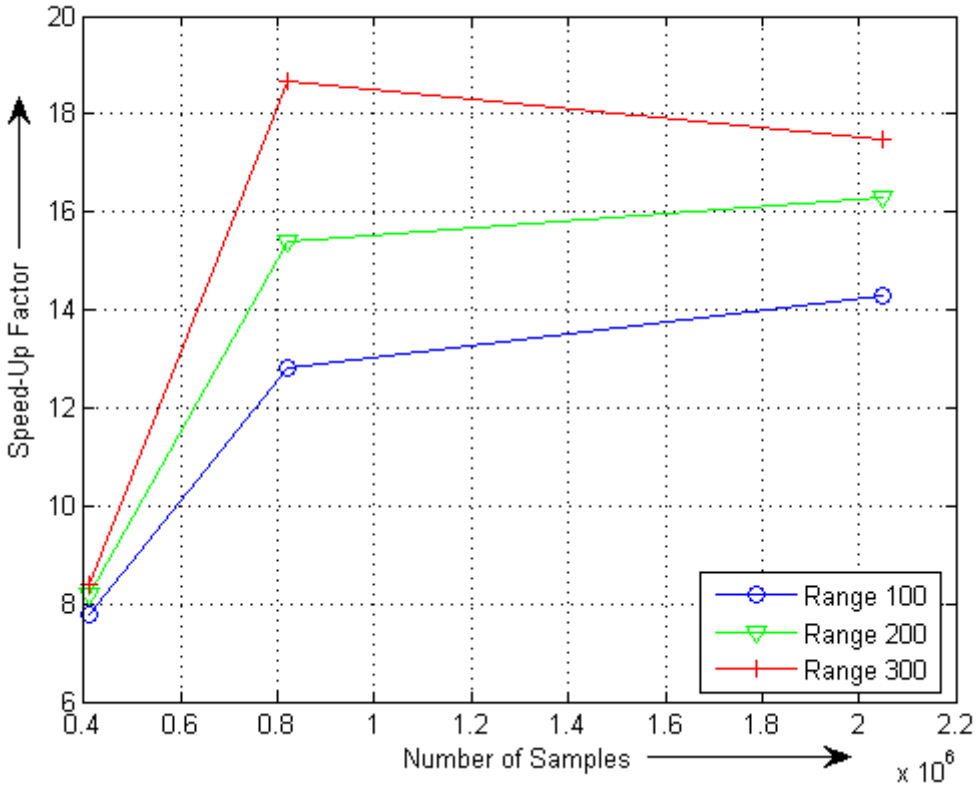

Figure 5.2: Variation of Speed-Up Factor for GPU-ARD with data size for different range bins

the program was executed in GTX 480 FX with a global memory of 1.5 GB (Appendix A). The program when tested in Tesla M1060 GPU [16], the speed-factor reduction at 2048K Sample data size was eliminated due to the higher global memory of 4 GB [16] .

5.2.2 Conclusions from results of GPU-ARD

The conclusion we can derive from the speed-up factor variation graph in Figure 4.6 is focused on the speed-up factor curve for Range 300. ARD processing for 300 Range-bins gives an effective range of 210 km considering a range resolution [57] of 721 m per range-bin. The effective range of 210 km is sufficient for the range coverage of the present system. The present ARD implementation on a single GPU has the maximum speed-up factor optimised at 300 range-bins. Hence the ARD-GPU implementation, highlighted by a speed-up factor of 18.67X significantly accelerates the PMR signal processing chain.

5.3 GPU-NLMS

5.3.1 Factors affecting Performance of GPU-NLMS

NLMS algorithm is introduced in Chapter 3, as a computationally efficient clutter cancellation algorithm. In this section the performance of the GPU implementation of NLMS termed as GPU-NLMS is studied with reference to the processing time comparison graph in Figure 5.3, and the speed-up factor variation graph in Figure 5.4.

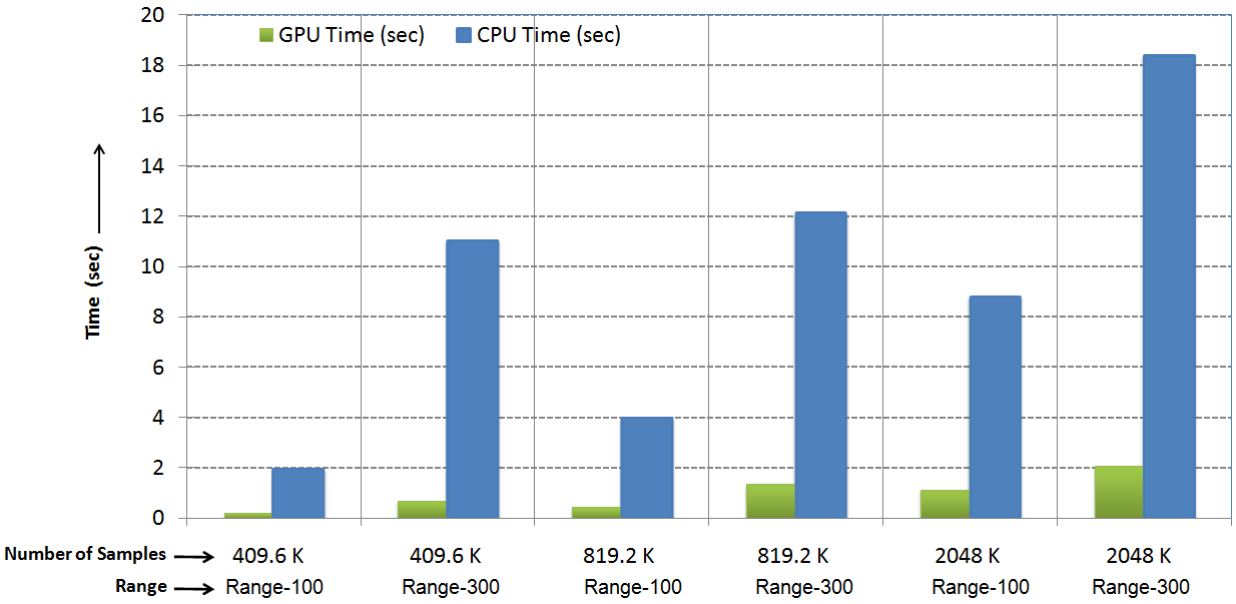

Figure 5.3: NLMS Time Consumption-CPU Vs GPU

Figure 5.4 reveals that the speed-up factor does not vary much in the case of NLMS with increase in data size. The reason for this can be better explained with reference to the implementation diagram already discussed in Figure 3.3. The kernels run with the number of parallel threads equal to the order of the filter which is turn is equal to the number of range-bins. The implementation is rather independent of the the data size due to the structure of the filtering model [25] shown in Figure 3.2. It is observed that the speed-up factor increases with the increase in number of range-bins, but not significantly. The reason is that the the number of range bins is only increased by 100 range-bins, for each set of observation, and this is not a considerable increase in the number of parallel threads. In short, the inherent algorithm structure of NLMS, limits maximum utilisation of the GPU cores.

The graph shown in Figure 5.4 showing a maximum speed-up of 9.2 for the curve for Range 300 followed by range 200 and range 100 provides evidence that the acceleration is more dependent on range-bins than data size. The reduction of speed-up factor with data size for Range 300 and Range 200 is due to the influence of the sequential part of NLMS algorithm.

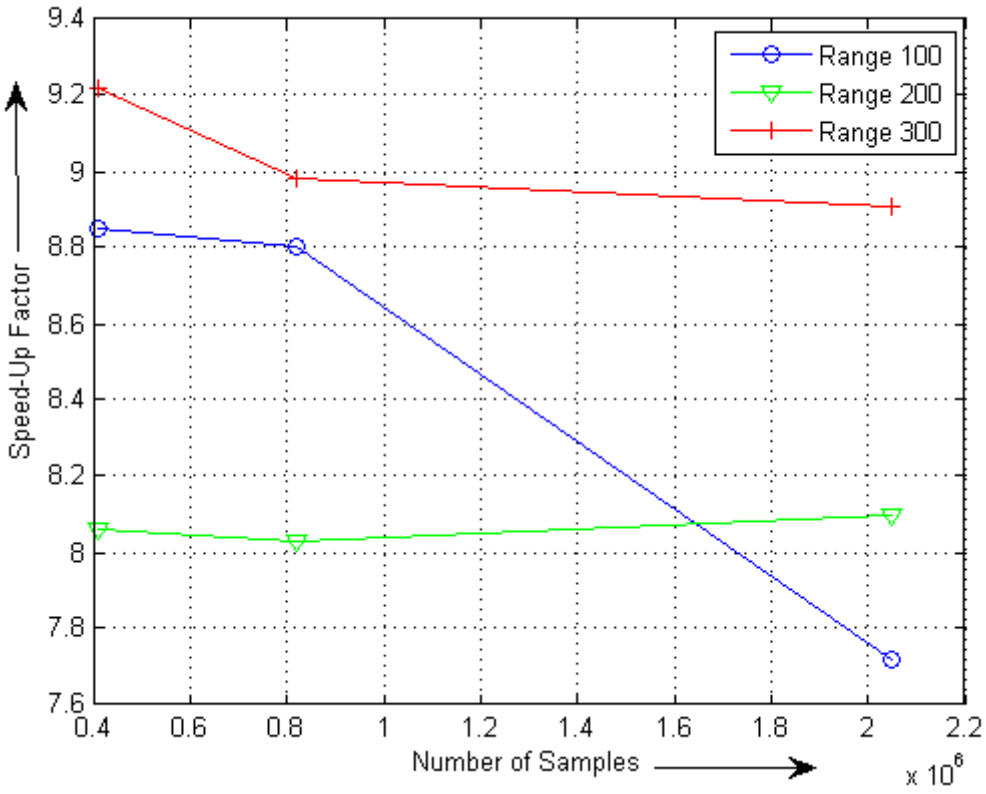

Figure 5.4: Variation of Speed-Up Factor for GPU-NLMS with data size for different range bins

5.3.2 Conclusions from results of GPU-NLMS

The actual execution time of NLMS algorithm states that the algorithm is computationally fast. Though the speed-up factor is not significant, the effective time for clutter cancellation taken by NLMS algorithm enables near real-time processing of PMR radar signals in a single GPU platform. This fact is supported by the the GPU execution time of 0.68 seconds for 1 second of observation for 300 range bins. When combined with ARD processing on single-GPU platform, the total time of execution will be 3.1 times real time processing.

5.4 GPU-ECA

5.4.1 Factors affecting Performance of GPU-ECA

The GPU implementation of ECA, with all its GPU subroutines is one of the most important part of this research project and the implementation was discussed in detail in Chapter 4. This Section describes the computational performance of the GPU implementation of ECA termed as GPU-ECA, with reference to Figure 5.5 and Figure 5.6.

GPU implementation of ECA was able to achieve better performance gain in terms of the speed-up factor than NLMS algorithm. The reason for this performance gain is that the inherent batch

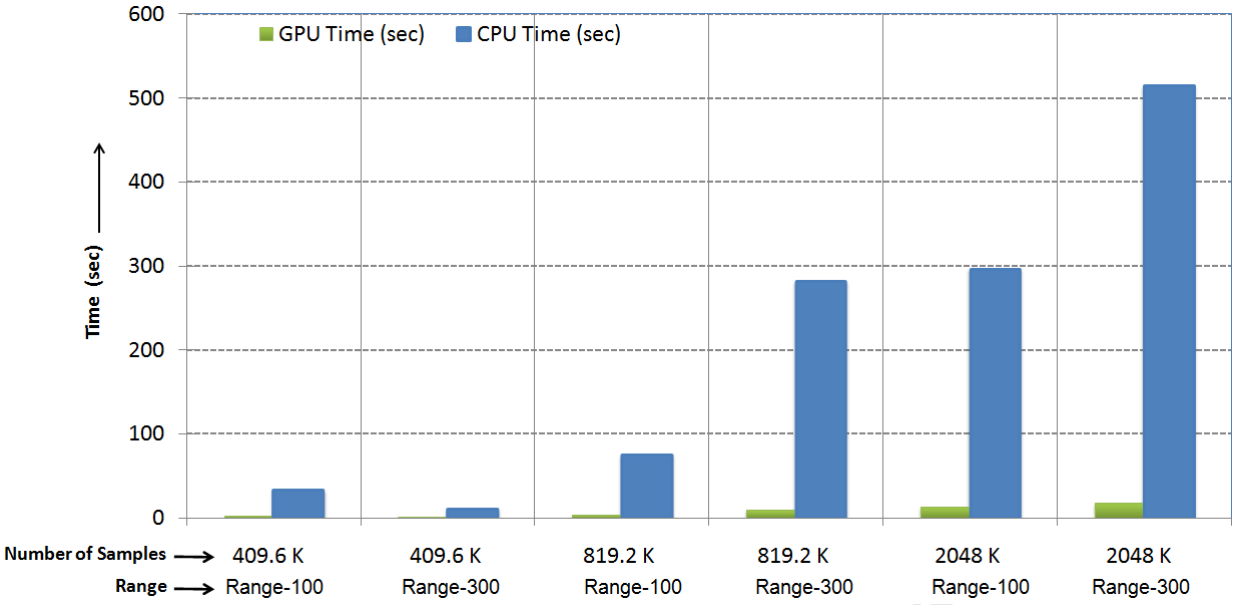

Figure 5.5: ECA Time Consumption-CPU Vs GPU.

process[25] nature of the ECA, fits perfectly into the parallel architecture of GPU. From Figure 5.5, its is observed that the speed-up factor increase proportional to the data size and the number of range-bins. This direct proportionality between data size and speed-up can be further explained, with reference to Figure 4.3. When processing is done for one second of observation, the data can be split at multi-processor level [46] exploiting all the parallel threads of GPU. When the number of range-bins increase, the calculation of multiple matrix product in subsection 4.6.2 and matrix inversion in Section 4.7, also achieves improved speed-up factor, further providing support for the computational performance of ECA.

5.4.2 Conclusions from results of GPU-ECA

Figure 5.6 illustrates speedup factor variation in detail. The implementation was able to achieve a speed-up factor of 27.9X for number of range-bins equal to 320 and for 2048K Samples of data (equivalent to 5 seconds of observation at capture frequency of 409.6 KHz). But ARD plot for 5 seconds of observation is not practical for a continuous air surveillance system. Processing an ARD plot from clutter cancelled data for 1 second or 2 seconds of observation is better suited for the present PMR system. Hence a speed-up factor between 25X and 27X can be expected from the ECA implementation in a single-GPU platform. When ECA is performed for 320 range-bins, an effective range of 224 km can be achieved with consideration to the range resolution [57] of 721m /range bin. This clutter cancelled captured data up to, 224 km is passed for ARD processing.

The effective range of ARD plot should be less than or equal to the range of clutter cancelled data. For example, when ARD is plotted for 300 range-bins yielding an effective range of 210 km, the range of clutter cancelled data should be greater than or equal to 210 km. Therefore the ARD plot will consist only clutter cancelled data. This proportion between extent of clutter cancellation and

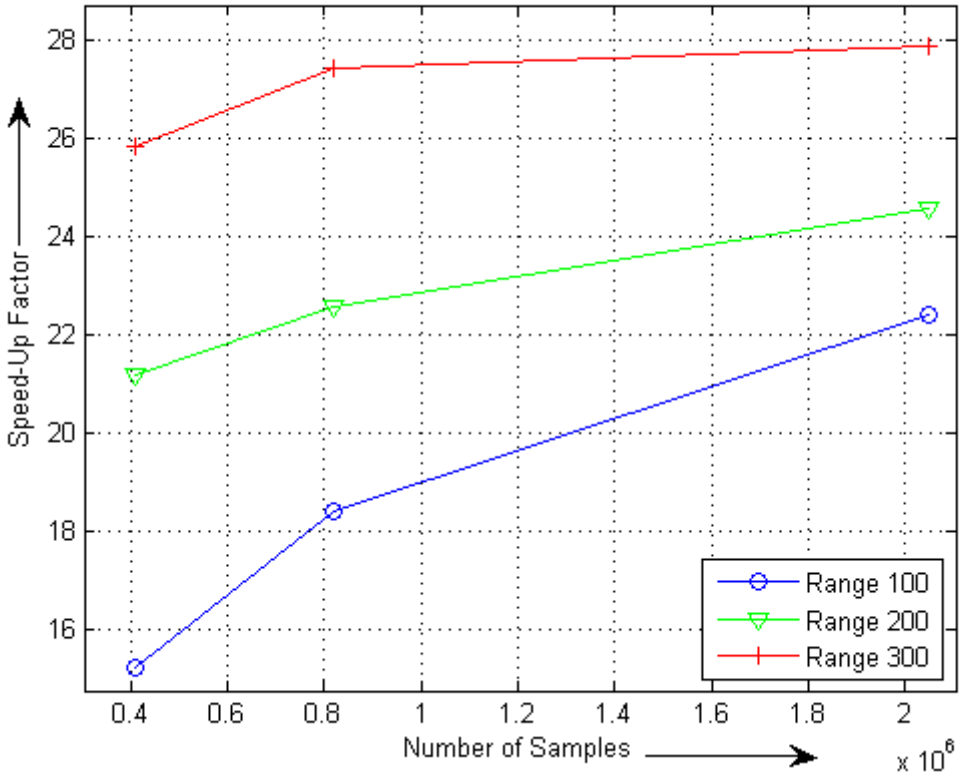

Figure 5.6: Variation of Speed-up factor for GPU-ECA with data size for different range bins

range of the ARD plot is very important, when the data is viewed in the streamline ARD plotter designed at RRSG by Craig Tong [55].

5.5 ECA vs NLMS

Figure 5.7 shows the time consumption of GPU processing for the two algorithms in a percentile scale. It is observed that NLMS algorithm is nearly 10 times faster than the ECA algorithm. Though the speed-up factor of NLMS from Figure 5.4 was less than the speed factor of ECA in Figure 5.6, test on the actual run-time illustrated in Figure 5.7 proves NLMS is computationally faster than ECA.

When the processing is done for data size of 819.2 K samples which is equivalent to 2 seconds of observation for range bins =300, NLMS algorithm is approximately 10 times more faster than ECA. But the choice of an algorithm for a radar system is based more on the efficiency of the algorithm than the computational performance. The efficiency of the two algorithm is discussed in the next section with reference to tests and results. The computational performance already discussed is combined with the algorithm efficiency test to draw a conclusion about the preferred algorithm in Section 5.9.

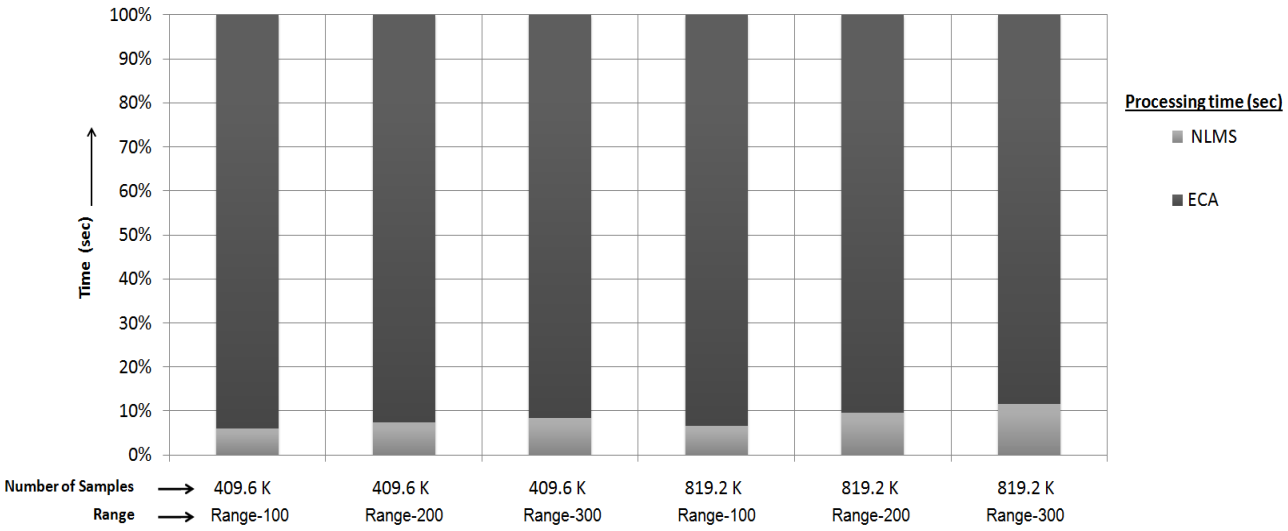

Figure 5.7: GPU Processing Time : ECA vs NLMS

5.6 Algorithm Efficiency Tests

The first section of this chapter was dedicated to the tests and results concerned with the computational characteristics of the implementation. In this section, the efficiency of the algorithm will be tested with both simulated and real data from various sources of illumination. The clutter cancellation efficiency of both the algorithms analysed using Signal to Clutter Ratio (SCR) measurements, for both short range and long range targets, are studied in detail in the following subsections.

5.6.1 Clutter Cancellation

Stationary objects in the surveillance area have zero Doppler frequency and has hence fall in the zero-Doppler axis in the ARD plot and constitute clutter. These unwanted signals have to be eliminated from observed data. Figure 5.8 shows an ARD plot when the processing is done without any clutter cancellation algorithm. The zero-Doppler axis, evident from the data cursor values marked in the Figure 5.8 constitutes of clutter data. The high value of clutter masks the targets in the Range-Doppler plane. The need for clutter cancellation arises at this point.

Figure 5.9 is the ARD plot obtained after clutter cancellation using NLMS algorithm. It is evident that the clutter at zero-Doppler have reduced considerably in short range, though total elimination of clutter is only observed after a certain number of range-bins. The reason for this is the time taken for filter to converge for a particular order of the filter. The reduction in clutter and multipath effect, have resulted in detection of a short range target at nearly 22 km though it is slightly masked by strong clutter which is also marked in Figure 5.9.

It is observed from Figure 5.9, that the Range-Doppler plane is almost flat for Range>100 km, which means that the NLMS algorithm was not successful in detecting long range targets. Improving the adaptivity of the filter by trial and error method can solve this problem, but that approach is not a

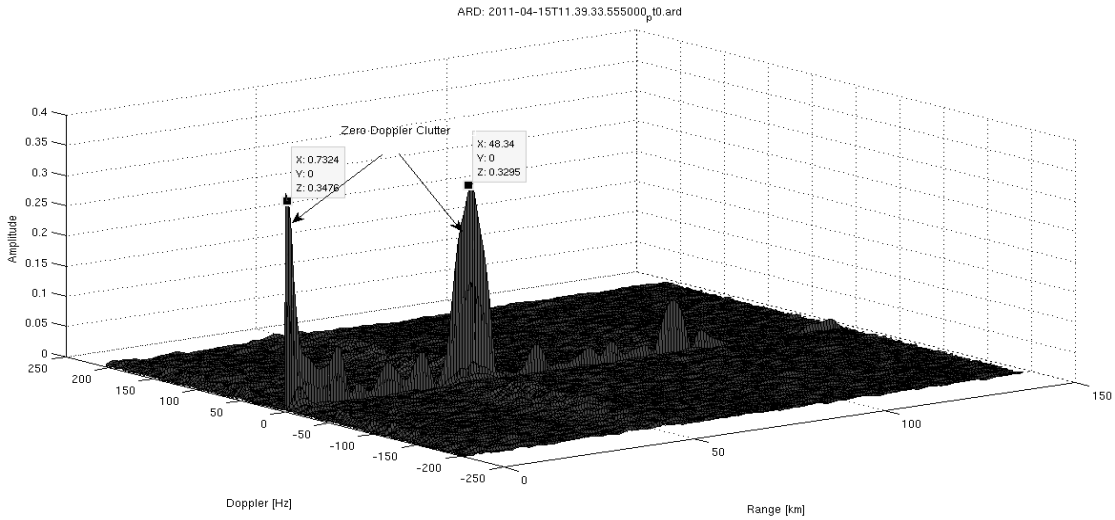

Figure 5.8: ARD 3D plot without cutter and DPI cancellation

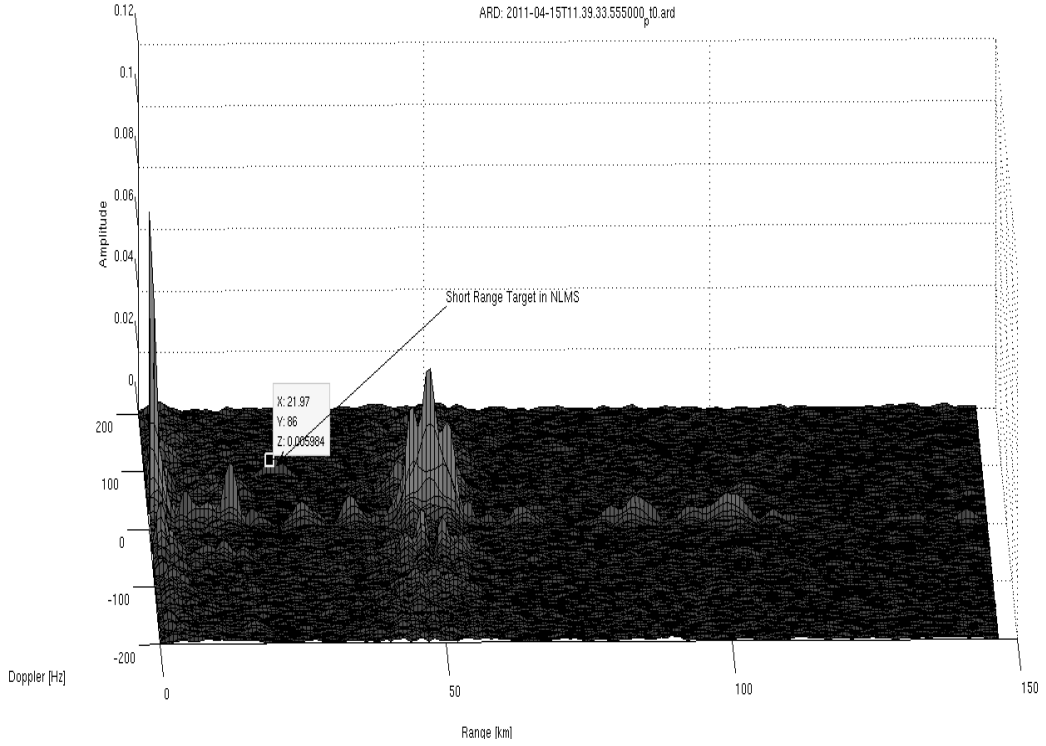

Figure 5.9: ARD-3D plot with NLMS clutter cancellation

reliable solution.

Figure 5.10 is the ARD plot obtained after ECA clutter cancellation. As it is evident from the computational tests on ECA in Section 5.5, the process took nearly 10X more time than NLMS, but the results were better with respect to zero-Doppler clutter cancellation and long range target detection. In Figure 5.11, target at 120km (range evident from X axis of data cursor) is detected together with clear short range detection. It is also observed that the clutter signal is almost fully eliminated in the zero Doppler axis.

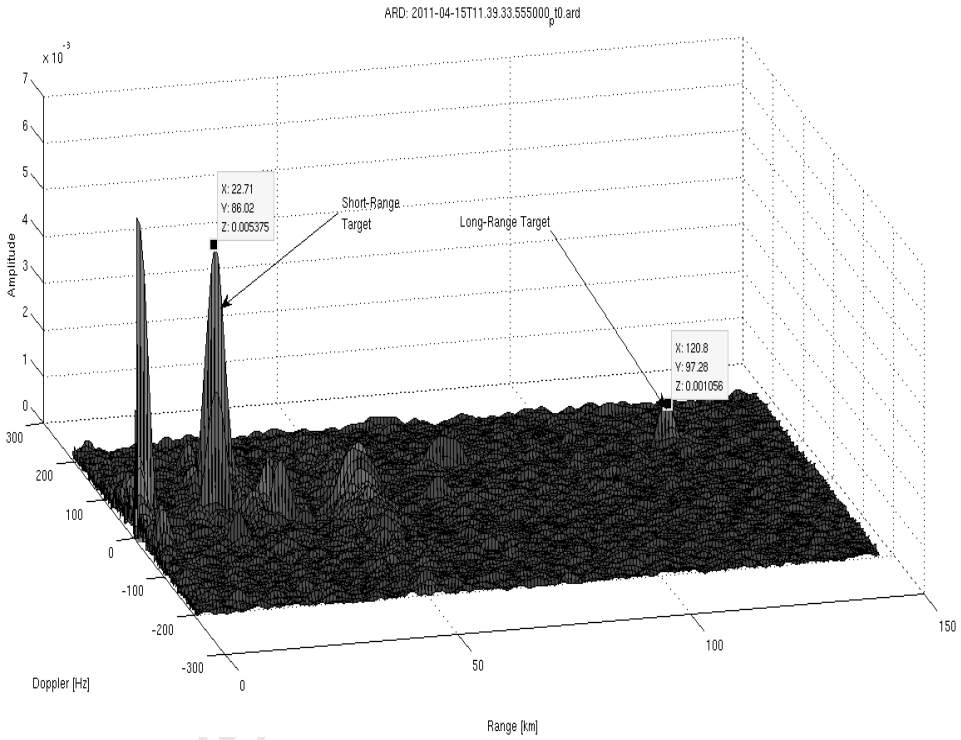

Figure 5.10: ARD-3D Plot with ECA Clutter Cancellation

5.6.2 Signal to Clutter Ratio ECA vs NLMS

Signal to Clutter Ratio (SCR) [54] is a performance parameter to calculate the efficiency of the algorithm in detecting moving targets in the presence of clutter environment. SCR is calculated as the ratio of the amplitude of the target signal, to the mean amplitude of the surrounding cells called guard cells. In comparing algorithms, guard cells are chosen based on assigning a threshold value to them. The calculation of SCR for short range and long range target on ARD plot obtained after ECA and NLMS clutter cancellation is given in Table 5.15.1. SCR, defined in Equation 5.1 is used for Table 5.1

$$SCR = \frac{A_T}{\frac{1}{n}\sum_{i=1}^{n} A_{Gi}} \tag{5.1}$$

where,

n is the total number of guard cells.

A_T is the amplitude of the target.

A_{Gi} is the amplitude of the i^{th}guard cells with i varying from 1 to n.

In Figure 5.11, the target at nearly 22 km from surveillance antenna which is comparatively short-range, is analysed for SCR calculation. The amplitude of the target response is marked using data cursor in Figure 5.11. The same value is entered in Table 5.1. The guard cells are enclosed in the green coloured dashed outline in Figure 5.11. The mean amplitude of the guard cells is calculated and is entered in Table 5.1. SCR is then calculated using Equation 5.1.

SCR is calculated using the same methodology for short and long range detection(~120 km) for both ECA and NLMS algorithms and the corresponding values are entered in Table 5.1 .

SCR Parameters	Short-Range (Target at 22km)		Long-Range (Target at 120km)	
	NLMS	ECA	NLMS	ECA
Amplitude of Target	0.006488	0.005375	0.001657	0.001108
Mean Amplitude of guard cells	0.002086	0.001693	0.0016097	0.000492
Signal to Clutter Ratio	3.11	3.15	1.02	2.24

Table 5.1: SCR Calculation: ECA vs NLMS for Short-Range (~22km) and Long-Range (~120km) Target

Conclusion about SCR from Table 5.1 can be split into short range and long range targets.

SCR for Short-Range Target

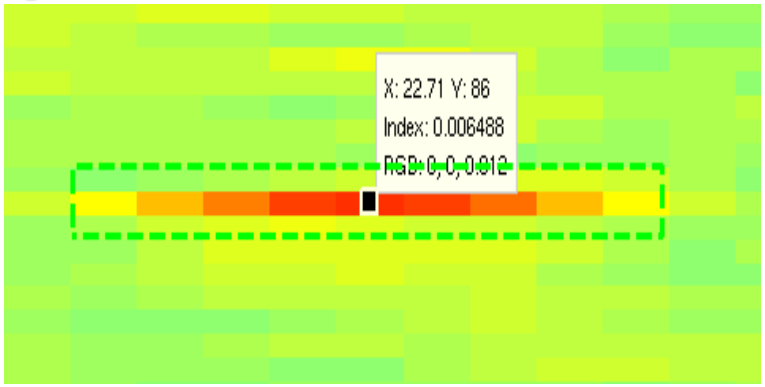

Figure 5.11: SCR Calculation for Short-Range Target detected in NLMS

The target observed is at nearly 22km from the surveillance antenna. The target is detected using both the algorithms. Calculation of SCR reveals that both the algorithms were able to eliminate

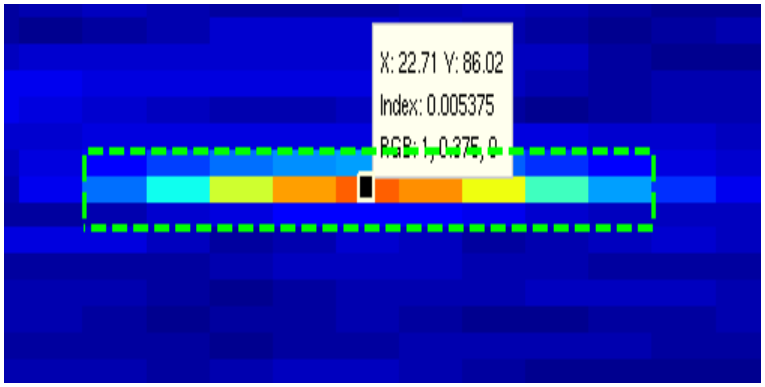

Figure 5.12: SCR Calculation for Short-Range Target detected in ECA

the masking of clutter efficiently, however ECA has a slight better performance with higher value of SCR. NLMS algorithm is hence found efficient and comparable to ECA in short range target detection.

SCR for Long-Range Target

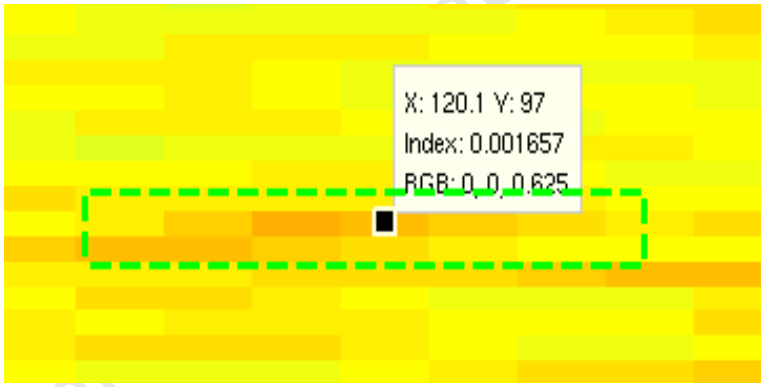

Figure 5.13: SCR Calculation for Long-Range Target detected in NLMS

The target under observation is at nearly 120.1 km and hence considered long-range. Figure 5.13 and Figure 5.14 is used for SCR analysis on long-range targets. The target was detected only in ECA. However the SCR analysis on ARD plot for NLMS provides evidence that NLMS is not suitable for long-range target detection. The SCR value from Table 5.1 is nearly equal to 1 which means, the target is strongly masked by clutter, or in other words the target and clutter are not distinguishable in the ARD plot. SCR analysis for ECA algorithm on the same target give promising result with SCR equal to 2.24 which means that the amplitude of the target response is 2.24 times greater than clutter signal. This makes ECA as a better algorithm for PMR systems in target detection especially when the processing stage is pipelined with CFAR detection in future. When CFAR detection is performed on the ARD data, depending upon the threshold set for false detection, responses with amplitude below above the threshold level will be only considered as targets. To meet this criteria, the target response should be sufficiently higher than surrounding clutter response.

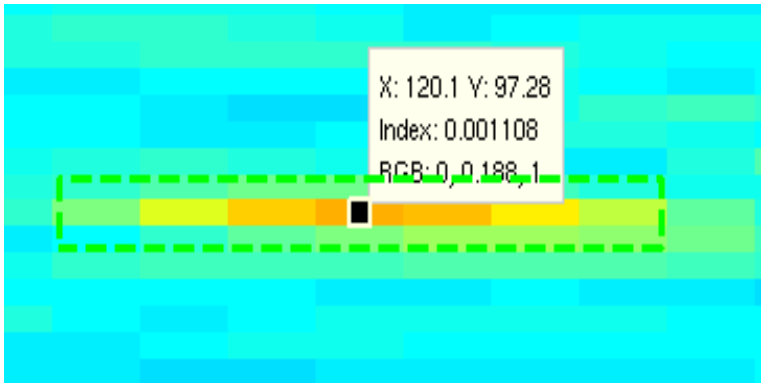

Figure 5.14: SCR Calculation for Long-Range Target detected in ECA

5.7 Test deployment result

Testing of the program was done in simulated data using Flexible Extensible Radar Simulator (FERS) [13] and the second phase of testing was done in real world scenario.

5.7.1 Testing using Simulated data

FERS [13] simulation was done for different target sizes- Big,Normal and Small. Figure 5.15 shows the ARD plotted for target of smaller cross-section. The effect of ECA, cancelling the clutter at zero-Doppler is visible in all the ARD plots.

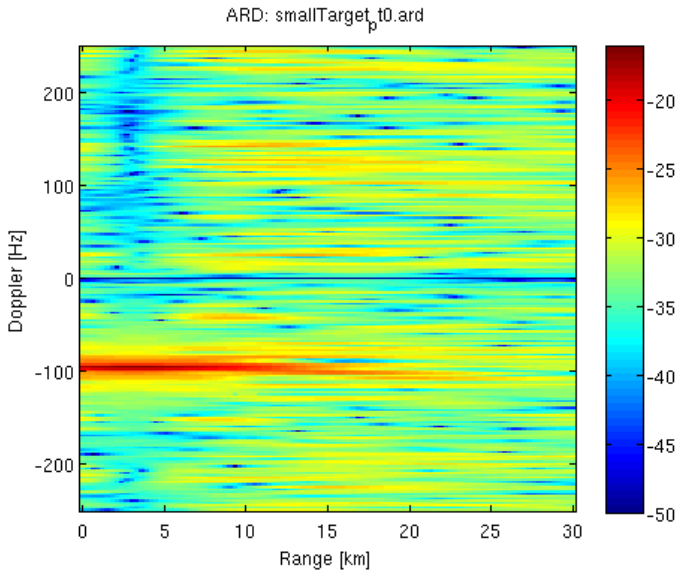

Figure 5.15: ARD Plot on FERS data on small target cross-section

Testing the program on FERS [13] simulation gave good results and motivated for the test deployment of the system in real world scenario.

5.7.2 Real-world Deployment

DVB-T data from Pisa

Figure 5.16 showed target detection from real-world DVB-T data from Pisa. Figure 3.5 was processed using NLMS algorithm. Figure 5.16 shows ARD plot obtained after ECA clutter cancellation for the same data. The target at (55,-350) and the clutter cancellation at zero-Doppler is clearly illustrated . Figure 5.16 provides the proof for the data independent nature of the ECA algorithm.

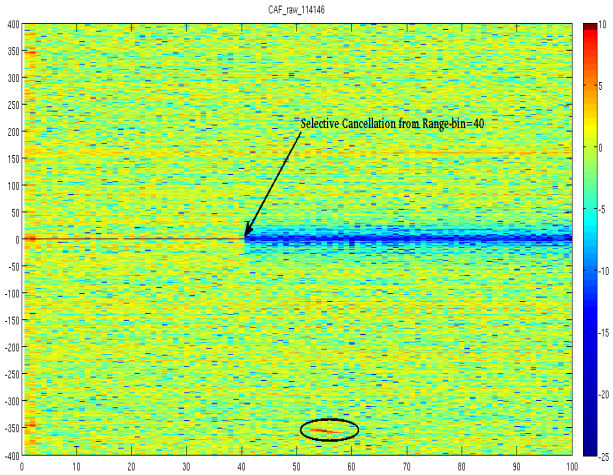

Figure 5.16: ARD Plot illustrating selective cancellation of ECA on DVB-T data. Target at (55,-350) encircled.

Figure 5.16 also highlights one of the additional advantages of ECA, the selective cancellation. It is observed that the clutter cancellation begins only from range 40 and then spans over the entire range. A minor editing in the cancellation algorithm can enable ECA to achieve selective cancellation so that area of surveillance can be limited to a certain range for specific application. The same property can be used for selective cancellation of unwanted targets. But the latter results in processing overhead and hence recommended only for specific uses.

FM data from Cape Town International Airport

Figure 5.17 is photograph from the test deployment of the PMR radar system at Tygerberg in Western Cape on December 15^{th} 2010. The reference and the surveillance antenna are marked in the picture. Data captures was sent to the radar lab at UCT using High-Speed Downlink Packet Access (HSDPA) link.

The target at (5,-10) in Figure 5.18 is a landing flight. The Doppler and range translation of the flight can be seen in the compound ARD plot in Figure 5.19 . The Compound ARD [55] is obtained by overlaying 1 second ARD frames over a 40 second observation time. The results were compared to the actual flight path obtained from a DBS transponder on the aircraft and a commercial receiver. From Figure 5.19 it is evident that the actual flight and the calculated flight are identical.

Figure 5.17: Photograph showing antenna position and coverage area from Trial 1

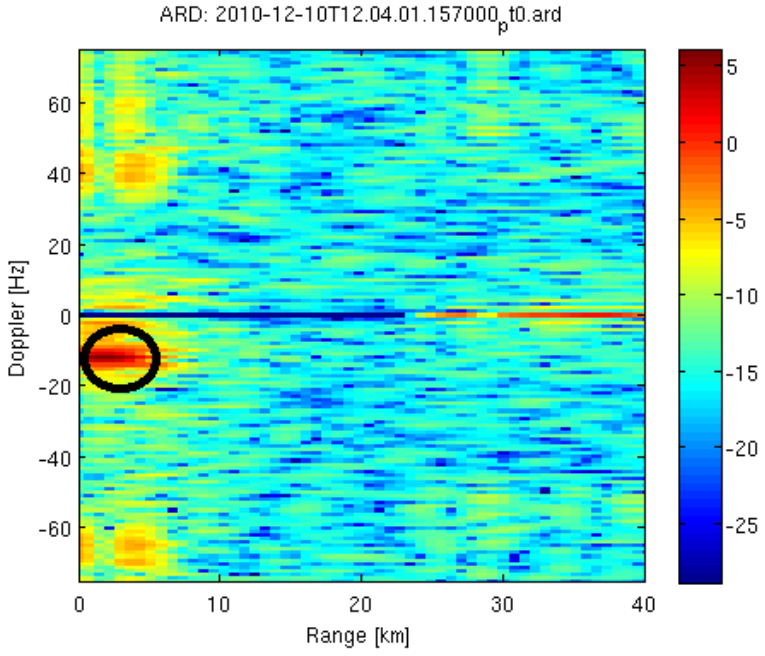

Figure 5.18: ARD from Test-deployment1-ECA

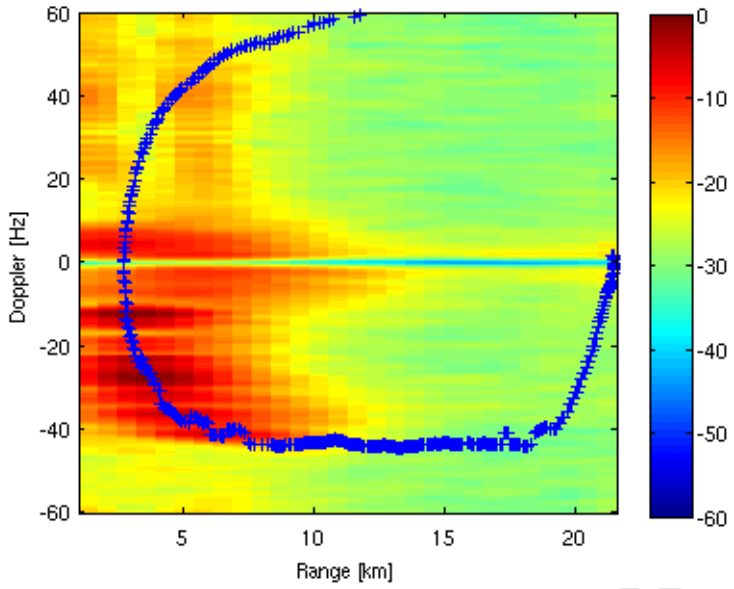

Figure 5.19: Compound ARD from Test-deployment1-ECA [55]

In the first deployment, target detection was obtained only at short range. But the second test deployment on April 15^{th} 2011, gave more evidence to prove the expected range and the efficiency of ECA algorithm. Figure 5.20 shows multiple target detections from second phase deployment.

In Figure 5.20, clutter cancellations done using ECA and in the ARD plot, both short-range and long-range targets are visible. Figure 5.11 and Figure 5.13 used for SCR analysis shows the output of NLMS processing for the same data set for short and long range. Targets at short-range were visible using NLMS, but long range target was not detected. Adjusting the input parameter μ_{NLMS} can improve the adaptivity of the filter for long range detection. But for an uninterrupted robust air traffic surveillance scenario, this is not practical.

5.8 Validation of processing with CPU processing

The results from the GPU processing of capture data from test deployment were compared to the results from CPU processing of the same data. This is a vital procedure to ensure the correctness of the processing and thereby validation of this research project. Figure 5.20 and Figure 5.21 are output ARDs from GPU and CPU respectively for the same capture data.

The processing is done for 4 seconds of observation for 1200 Doppler bins centres at 0 Hz for 395 range bins. 4 seconds of observation is equivalent to a data size of 1638.4 K samples of complex data. ECA algorithm is used for DPI and clutter cancellation. It is observed that the two ARD plots in Figure 5.20 and Figure 5.21 are very similar in terms of positions of the targets (encircled) in range and Doppler axis. The amplitude of the reflected signal from the target is also similar which is evident by comparison of the colour bar. The plots are obtained in the streamline plotter named ARDView [55] designed for the PMR project. The header to the '.ard' [55] file which highlights the

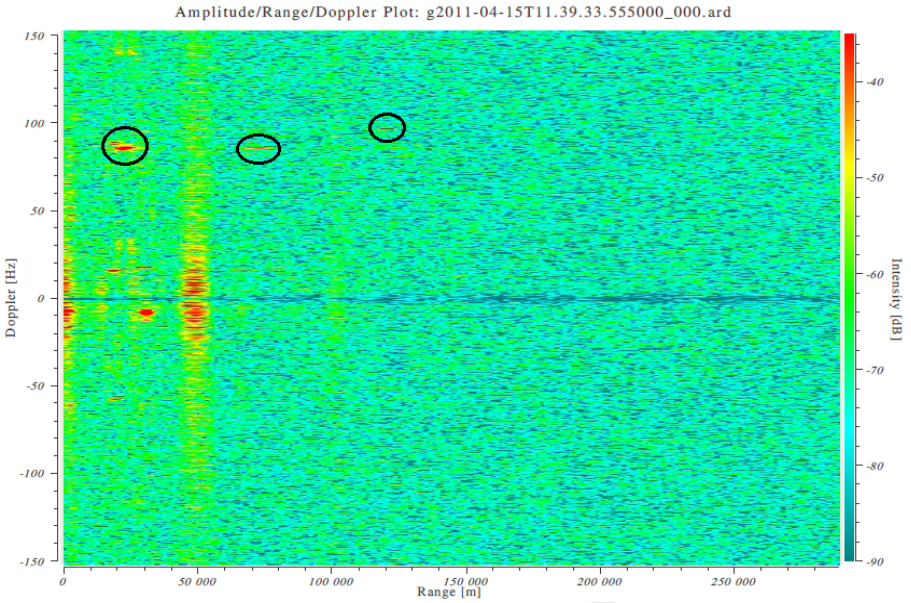

Figure 5.20: Long range Multiple target ARD from Test-deployment II-ECA on GPU (The title of the plot prefixed by 'g' refers to GPU Processed ARD plot)

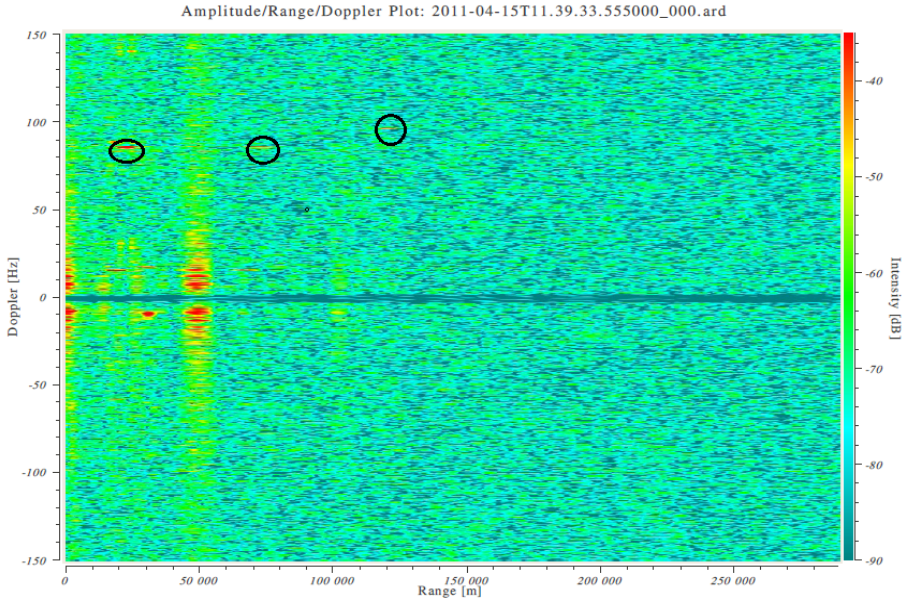

Figure 5.21: ARD from Test-deployment II-ECA processed on CPU

processing and signal parameters are set exactly identical so that the comparison between the CPU and the GPU ARD plot can be considered as a reliable validation of the PMR signal processing in GPU platform.

5.9 Interpretation of Tests and Results

- Computational Efficiency
  - NLMS algorithm is computationally faster than ECA with reference to results illustrated in Figure 5.7. But the GPU implementation of ECA with reference to the graph in Figure 5.6 provided evidence that ECA algorithm is well suited for parallel computation than NLMS. Though the processing time of GPU-ECA was larger than GPU-NLMS, the batch processing nature of the ECA is promising for multi-GPU implementation. The reason for this difference in effective speed-up factor is an implication of the Amdhal's law [37]. The sequential processes in NLMS algorithm reduces the overall speed-up factor whereas in ECA, the major portion of algorithm is parallelisable and hence achieves a higher speed-up factor. Thus in a multi-GPU environment, GPU-ECA can compete with GPU-NLMS in computational efficiency enabling real time-processing.
- SCR Analysis
  - The SCR analysis in Section 5.6 with the results in Table 5.1 provided evidence for the enhanced clutter cancellation efficiency of ECA. Both ECA and NLMS clutter cancellation gave comparable SCR values for short range targets. But for long range targets (>100 km), NLMS algorithm exhibited relatively poor performance and the targets were not detected. ECA on the on the other hand was able to detect targets with good SCR.
- Source independent nature and Selective cancellation.
  - ECA was used for clutter cancellation for both FM and DVB-T based sources of illumination without any change in input parameters. NLMS on the other hand have to be provided with suitable adaption factor as the input variable by trial and error method with change in the source of illumination. This aspect makes ECA more reliable and robust than NLMS algorithm.
  - The selective clutter cancellation ability of ECA illustrated in Figure 5.16 enables limiting of clutter cancellation to preferred range of interest. This feature is not achievable with the NLMS algorithm and can be considered as an additional supporting factor for ECA.

5.10 Complete System Performance

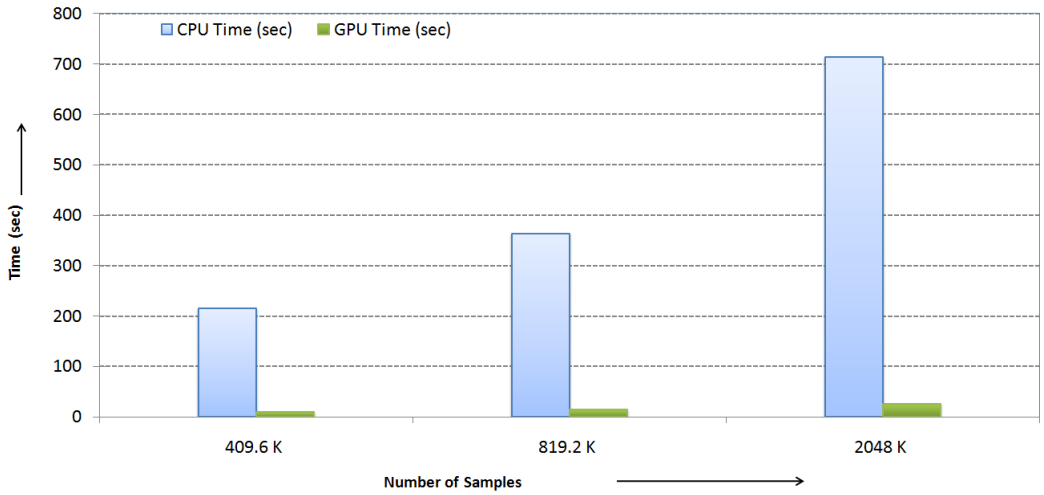

Figure 5.22: Complete system run-time comparison between CPU and GPU.(The processing is done using ECA with cancellation for 300 Range bins, Doppler bins=3 and ARD plot for 300 Range bins and 1201 Doppler bins centred at 0 Hz.)

The complete system performance with respect to the total run-time comparison between CPU and GPU is shown in Figure 5.22 and the overall speed-up factor is shown in Figure 5.23. The previous section selected ECA as the preferred algorithm for PMR signal processing. In Figure 5.22, ECA is used together with ARD processing for the time comparison between CPU and GPU for 300 range bins. It is observed that the run-time for GPU implementation is much faster than the CPU implementation and the time difference of processing between the two platforms increases with increase in the data size. This point can be better observed in the overall speed factor variation graph in Figure 5.23.

In Figure 5.23, PMR signal processing using ECA as cancellation algorithm is designated as PMR(ECA) and processing using NLMS algorithm is designated as PMR (NLMS). NLMS algorithm is included since the algorithm was found efficient for DVB-T based target detection as evident from Section 3.7. PMR signal processing was able to achieve a total speed-up factor of 29.43X in a single GPU platform using ECA and 16.39X using NLMS cancellation algorithms respectively. The performance speed-up achieved is promising for implementation using multiple GPUs after integrating the system with a hyper threaded CPU platform which is a part of future work in this field and will be discussed in Chapter 6.

5.11 Conclusion

This Chapter gave a detailed illustration of the test and results of the GPU-ARD, GPU-NLMS and GPU-ECA with respect to various performance metrics. The two algorithms were compared with respect to their computational performance. The ECA was able to achieve better speed-up than

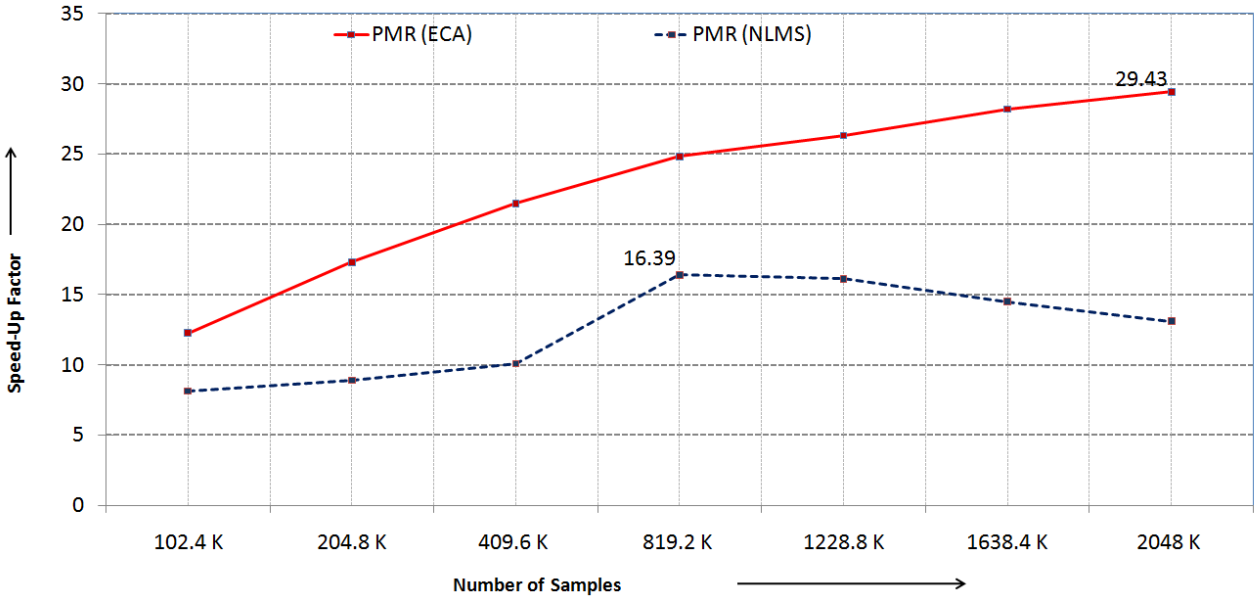

Figure 5.23: Overall Speed-Up Factor for the PMR system. (The processing is done using ECA with cancellation for 300 Range bins, Doppler bins=3 and ARD plot for 300 Range bins and 1201 Doppler bins centred at 0 Hz.)

NLMS which can be regarded as a direct implication of Amdhal's law [37] in multi-processor system. However when comparing the algorithms based on run-time, NLMS is much faster than ECA. The Chapter also compared the two algorithms based on their clutter cancellation efficiency using detailed SCR analysis for short and long range targets. Based on the test and results, ECA algorithm was selected as the preferred algorithm for PMR signal processing, though NLMS algorithm was found efficient for DVB-T based systems. The Chapter also validated the overall GPU implementation with the results from the CPU implementation by comparing ARD plots with the same input data. Finally, the Chapter illustrated the overall system performance in GPU platform with respect to the CPU platform. The Conclusions from the dissertation and leads for further improvement of the system in a multi-threaded multi-gpu heterogeneous platform is discussed in Chapter 6.

Chapter 6

Conclusions and Recommendations

6.1 Summary

This dissertation can be summarised into the following work in the GPU sub-system, that was completed as a part of the PMR project:

- Literature reviews of PMR system and GPGPU were presented. Literature on PMR is relatively short and is confined to the signal processing part. Extensive references about previous work and PMR properties were included in the literature review for further information. GPGPUs were discussed, with more details of CUDA SDK and its properties.
- The computationally intensive stages of PMR signal processing chain were identified and the compatibility of the algorithm nature for parallel processing was analysed.
- Matched Filtering/ARD Processing was modelled and implemented in GPU.
- Two distinct clutter cancellation algorithms- NLMS and ECA were modelled and implemented in GPU platform.
- NLMS and ECA were compared and benchmarked based on their computational efficiency and clutter cancellation efficiency with respect to SCR analysis.
- The performance analysis of individual sections and the complete systems was done. The GPU implementation was compared to CPU implementation with respect to computational and algorithm efficiency.
- Validation of the GPU processed results with reference to the CPU results was done.
- Details and results from the test deployment of the system were presented.

6.2 Conclusions

This dissertation provides evidence that GPUs are a promising platform for PMR signal processing, and in general for radar signal processing. The signal processing of the PMR system handles very large data sizes and the parallel nature of the algorithm fits ideally for the GPU processing.

GPU implementation of matched filtering, NLMS algorithm and ECA were discussed in detail with reference to flowcharts and program flow. The optimisation measures and the performance parameters were discussed in detail. The implementation was tested extensively according to standard performance metrics. The performance gain of the GPU implementation in terms of computational efficiency is compared to the CPU with respect to the speed-up factor and effective run-time. It is to be noted that the CPU code used for comparison can be well optimised by multi-threading. The effective speed-up achieved for the overall system in single-GPU platform supports for extending the project to a multi-GPU platform.

The dissertation also studied the clutter cancellation efficiency of both the algorithms based on SCR analysis. The SCR at short-range for NLMS and ECA were comparable, but when it comes to long-range detection ECA was found to be far more efficient than NLMS with a higher value of SCR. Testing of ECA with DVB-T data with successful target detection together with selective cancellation ability provides further supporting factor for ECA algorithm to use in the extended PMR system under design for different signal environment.The performance comparison of the two algorithms discussed in Chapter 5 gives evidence that ECA, though computationally more demanding than NLMS, is a better clutter cancellation algorithm for the PMR systems in GPU platform. The choice is made with consideration to the computational cost in a multi-gpu platform and the increased reliability and robustness that can be achieved for the overall system.

Details of the test deployment of the systems at Tygerberg in Western cape using FM data and in Pisa using DVB-T data and target detections using GPU processing were discussed in detail. Results form the test deployment provided evidence for the efficiency of GPU platforms for the PMR signal processing in a real-world scenario.

6.3 Recommendations for Future Work

The present implementation is now working as an offline processing code. The integration of the code to the streamline capture manager (under development) and pipelining the output of matched filtering to the real time plotter [55] and CFAR detector [54], which is under development, is part of the future work on this project.

This research work mainly focuses on the implementation of NLMS filter in the GPU platform. However there are other variation of filters like Gradient adaptive lattice (GAL),Least Square Lattice etc. which have to be tried in GPU platform for better results in terms of both clutter-cancellation efficiency and processing time. Alternate algorithms for ARD processing using blocked FFT approach

also provides scope for future work, though it is recommended to use GPU platform only for large data sizes, and hence blocked FFT approach only becomes applicable for longer observations times.

Recommendations for upgrading the implementation to a multi-GPU platform also include optimisation of the CPU platform to deliver maximum performance. Active research on parallel computation using GPUs in radar technology is being initiated at many advanced research agencies in the World. Parallel computation for Synthetic Aperture Radar (SAR) [54] systems at Fraunhofer Institute for High Frequency Physics and Radar Techniques (FHR) [4] is one such research project. Radar signal processing consists of a combination of sequential and parallel computing phases. Hence the preferred platform for radar signal processing is a highly optimised heterogeneous platform. The future work on this project should be the migration of the computing platform to a multi-threaded CPU, multi-GPU heterogeneous platform. An efficient Message Passing Interface (MPI) Center [14] have to developed when the scope of the project increases to the level of a distributed system. Finally, extensive testing and optimisation of the processing code, after integration to the fully automated PMR system under design, have to be performed before releasing the prototype of the PMR air surveillance system.

Appendix A

CPU-Specification and Software Environment

Software Environment	
Operating System	Ubuntu 10.04 (Lucid Lynx) LTS
gcc Compiler	4.3
CUDA Driver Version	3.0
CUDA Runtime Version	3.0

CPU Specification	
Manufacturer	AMD
Model	AMD Phenom(tm) II X4 955 Processor
Speed	3.2 GHz
Peak Processing Performance (PPP)	51.2 GFLOPS
RAM	16 GB DDR2 800MHz
Adjusted Peak Performance (APP)	15.33 WG
Cores per Processor	4 Unit(s)
Threads per Core	1 Unit(s)
Type	Quad-Core
Bus	HyperTransport
Maximum Speed	3.2 GHz
Maximum Power	176.61 W

Appendix B

Nvidia GTX 480 FX-Specification

GPU Engine Specification	
CUDA Cores	480
Graphics Clock (MHz)	700 MHz
Processor Clock (MHz)	1401 MHz
Texture Fill Rate (billion/sec)	42

Memory Specification	
Memory Clock (MHz)	1848
Standard Memory Configuration	1536 MB GDDR5
Memory Interface Width	384-bit
Memory Bandwidth (GB/sec)	177.4

Property	Value
CUDA Capability Major revision number	2
CUDA Capability Minor revision number	0
Number of multiprocessors	15
Total amount of constant memory	65536 bytes
Total amount of shared memory per block	49152 bytes
Total number of registers available per block	32768
Warp size	32
Maximum number of threads per block	1024
Maximum sizes of each dimension of a block	1024 x 1024 x 64
Maximum sizes of each dimension of a grid	65535 x 65535 x 1
Maximum memory pitch	2147483647 bytes
Texture alignment	512 bytes

Bibliography

[1] Nvidia's next generation cuda compute architecture: Fermi. Technical report, NVIDIA Corporation, 2009.

[2] *Tesla Data Center Solutions*. [Online] Available: http://www.nvidia.com/object/preconfigured-clusters.html, Accessed: February 15, 2011.

[3] *Gaussian Elimination*. [Online] Available: http://www.sosmath.com/matrix/system1/system1.html, Accessed: January 12, 2011.

[4] *Fraunhofer Institute for High Frequency Physics and Radar Techniques*. [Online] Available:http://www.fhr.fraunhofer.de/fhr/fhr_c628_f7_en.html, Accessed: May 15, 2011.

[5] *Soviet Radar in WW II*. [Online] Available: http://www.soviethammer.info/blog/519402-soviet-radar-in-ww-ii/, Accessed: November 1, 2010.

[6] *Automatically Tuned Linear Algebra Software (ATLAS)*. [Online] Available: http://math-atlas.sourceforge.net/, Accessed: October 20, 2010.

[7] *BLAS(Basic Linear Algebra Subprograms)*. [Online] : http://www.netlib.org/blas/, Accessed: October 30, 2010.

[8] *Nvidia GTX 480*. [Online] Available: http://www.nvidia.com/object/product_geforce_gtx_480_us.html, Accessed: September 5, 2010.

[9] A. Moreiga G. Krieger P. Dubois-Fernandez H. Cantalloube B. Vaizan M. Cherniakov T. Zeng P. Howland H.Griffiths C.Baker A. Moccia, M.D Errico and J. Sahr. *Bistatic Radar: Emerging Technology.* John Wiley & Sons Ltd, 2008.

[10] Ronald W. Scafer Alan V. Oppenheim. *Digital Signal Processing*. Prentice-Hall, 1975.

[11] D. M. Ritchie. B. W. Kernighan. *The C Programming Language*. Prentice Hall Inc, 1988.

[12] Constantine A. Balanis. *Antenna Theory: Analysis and Design*. John Wiley & Sons, 2005.

[13] Marc John Brooker. *The Design and Implementation of a Simulator for Multistatic Radar Systems*. Doctoral thesis, University of Cape Town - RRSG, June 2008.

[14] Maui High Performance Computing Center. *Message Passing Interface*. [Online] Available: http://www.mhpcc.edu/training/workshop/mpi/MAIN.html, Accessed: July 12, 2011.

[15] F. Colone. A multistage processing algorithm for disturbance removal and target detection in passive bistatic radar. In *IEEE Trans. On Aerospace and Electronic Systems*, volume 45, pages 698–721, 2009.

[16] Nvidia Corporation. [Online] Available: http://www.nvidia.com/, Accessed: February 5, 2011.

[17] Y. Lu M. Lesturgie D. Tan, H. Sun and H. Chan. Passive radar using global system for mobile communication signal: theory, implementation and measurements. In *Radar, Sonar and Navigation, IEE Proceeding*, volume 152, pages 116–123,, June 2005.

[18] S. Muller DH. Kuschel, J. Heckenbach and R. Appel. On the potentials of passive,multistatic, low frequency radars to counter stealth and detect low flying targets. pages 1–6, May 2008.

[19] Matt Ettus. *USRP User's and Developer's Guide*. Ettus Research LLC, Matt Ettus, Ettus Research LLC.

[20] Andreas Falkenberg. Method to calculate the inverse of a complex matrix using real matrix inversion. Technical report, Im Espenhagen 10 ; 51702 Bergneustadt, Germany.

[21] Michael Rice Fredric J. Harris, Chris Dick. Digital receivers and transmitters using polyphase filter banks for wireless communications. In *IEEE Transactions On Microwave Theory And Techniques*, volume 51, page 4, April 2003.

[22] H. Griffiths. Bistatic and multistatic radar. Technical report, University College London.

[23] H. Griffiths and N. Long. Television-based bistatic radar. In *Communications, Radar and Signal Processing*, volume 133,, pages 649–657. IEE Proceedings, December 1986.

[24] G.Bishop G.Welch. An introduction to the kalman filter, 2006.

[25] Simon Haykin. *Adaptive Filter Theory*. Prentice Hall, fourth edition, 2002.

[26] Francois Sebastiaan Heunis. Passive coherent location radar using software-defined radio techniques. Master's thesis, University of Cape Town, March 2010.

[27] P. Howland. Editorial: Passive radar systems. In *Radar, Sonar and Navigation, IEE Proceedings*, volume 152, pages 105–106, June 2005.

[28] P. Hudson. Passive multistatic radars in anti-stealth air defence. Master's thesis, Canadian Forces College, 2003.

[29] Michael Raymond Inggs, Yoann Paichard, and Gunther Erich Lange. Networked pcl system. In *Proceedings of the 2010 Cognitive Systems with Interactive Sensors (COGIS 2010)*. IET United Kingdom, 2010.

[30] National Instruments. *PCI Express*. [Online] Available: www.ni.com/pciexpress/, Accessed: February 5, 2011.

[31] A. M. Cunningham L. Martin J. Baniak, G.Baker. *Silent sentry passive surveillance*. [Online] Available: www.blockyourid.com/g̃bpprorg/mil/radar/sentry.pdf, Accessed: December 1, 2010.

[32] M. Jackson. The geometry of bistatic radar systems. In *Communications, Radar and Signal Processing,IEE Proceedings*, volume 133, pages 604–612, December 1986.

[33] D. Luebke S. Green J.E. Stone J.C. Phillips J.D. Owens, M.Houston. Gpu computing. In *Proceedings of the IEEE*, volume 96, pages 879–899, May 2008.

[34] W. A. Wall K. A. Duke. A professional graphics controller. *IBM Systems Journal*, 24(1):14, 1985.

[35] K.Szumski. Real-time software implementation of passive radar. In *Proceedings of the 6th European Radar Conference*, pages 33 – 36, September 2009. ISBN 978-1-4244-4747-3. doi: Sept.302009-Oct.22009.

[36] M. Brooker M Inggs. Extensible simulator for waveform diversity testing. In *Waveform Diversity and Design Conference*, pages 273–277, 2009.

[37] M.R. Marty M.D. Hill. Amdahl's law in the multicore era. *IEEE Computer Society*, pages 33–38, 2008.

[38] S.G. Johnson M.Frigo. *FFTW User Manual*, 2009.

[39] Y. Paichard M.Inggs, G.Lange. A quantitative method for mono- and multistatic radar coverage area prediction. In *Proceedings of the 2010 IEEE Radar Conference*, May 2010.

[40] Ethan Mollick. Establishing moore's law. In *Annals of the History of Computing, IEEE*, volume 28, pages 62 – 75, September 2009.

[41] Norman Morrison, Richard Thomas Lord, and Michael Raymond Inggs. The gauss-newton algorithm in passive aircraft tracking using doppler and bearings. In *Proceedings of the IET International Conference on Radar Systems (RADAR 2007)*. Institution of Engineering and Technology, October 2007.

[42] B. T. Neale. Ch - the first operational radar. *The GEC Journal of Research*, 3(2):73–83, 1985.

[43] R. Nitzberg. *Radar signal processing and adaptive systems*. Artech House, 1999.

[44] *CUDA CUFFT Library Version 1.1*. NVIDIA, Santa Clara, CA, October 2007.

[45] *CUDA CUBLAS Library*. NVIDIA, Santa Clara, CA, March 2008.

[46] *NVIDIA CUDA Programming Guide*. NVIDIA, version 3.0 edition, February 2010.

[47] *Tuning CUDA Applications for Fermi Version 1.2*. NVIDIA, Santa Clara, CA, July 2010.

[48] *NVIDIA CUDA Reference Manual Version 3.0*. NVIDIA, Santa Clara, CA, February 2010.

[49] *NVIDIA CUDA Best Practices Guide 3.0*. NVIDIA Corporation, 2010.

[50] D. Maksimiuk P. Howland and G. Reitsma. Fm radio based bistatic radar. In *Radar, Sonar and Navigation, IEE Proceeding*, volume 152, pages 107–115,, June 2005.

[51] R.Lyons. *Understanding Digital Signal Processing*. Prentice Hall, 2004.

[52] Conrad Sanderson. *An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments*. NICTA, St Lucia, Australia, September 2010.

[53] Scott.C.Douglas. A family of normalized lms algorithms. In *IEEE Signal Processing Letters*, volume SPL-1, pages 49–51, 1994.

[54] Merrill Skolnik. *Introduction to Radar Systems*. MCGraw-Hill, 1980.

[55] Craig Tong, Michael Raymond Inggs, and Gunther Erich Lange. Processing design of a networked passive coherent location system. In *Proceedings of the 2011 IEEE Radar Conference*, May 2011.

[56] R.I. Wilkinson. *Short survey of Japanese Radar-1*. [Online] Available:http://dreadnoughtproject.org/friends/dickson/Japanese%20Radar%20Short%20Survey.pdf, Accessed: December 1, 2010.

[57] N.J. Willis. *'Bistatic radar' chapter 25 in Radar Handbook*. McGrawHill, second edition, 1990.